
Table of Contents

Introduction	1.1
Setup	1.2

OpenShift 201 - Pipeline your DevOps

Overview	2.1
GitFlow	2.2
Using OpenShift Templates	2.3
Helm Templates	2.4
Building OpenShift Templates	2.5
Jenkins Pipeline As Code	2.6
Jenkins Build Pipeline	2.7
Jenkins Pipeline Parallelization	2.8
HA DB Installation	2.9
DB Configuration	2.10
DB Backup and Restore	2.11
Failure Testing	2.12
Health Checks	2.13

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).

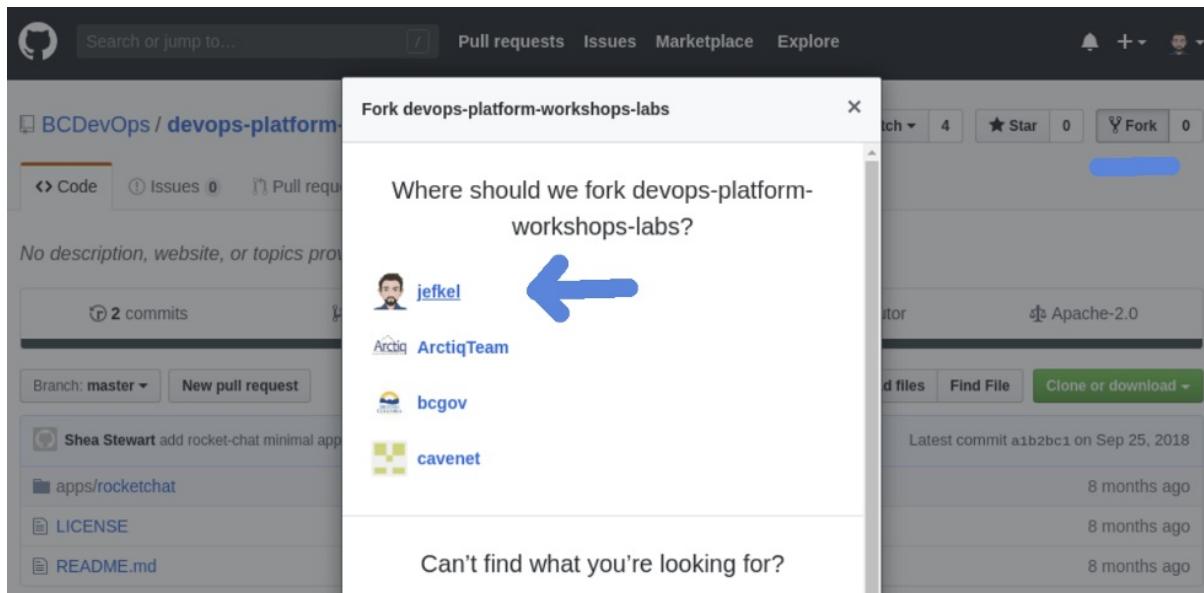
The content in the labs will get `slimmer` as labs progress. The goal here is that previous learnings will help you familiarize yourself with the Web Console or CLI utilities.

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
```

```

jkelly@jden:~/projects/devops-platform-workshops-labs
File Edit View Search Terminal Help
[jkelly@jden projects]$ git clone git@github.com:jefkel/devops-platform-workshops-labs.git
Cloning into 'devops-platform-workshops-labs'...
remote: Enumerating objects: 12, done.
remote: Total 12 (delta 0), reused 0 (delta 0), pack-reused 12
Receiving objects: 100% (12/12), 8.20 KiB | 8.20 MiB/s, done.
Resolving deltas: 100% (1/1), done.
[jkelly@jden projects]$ cd devops-platform-workshops-labs/
[jkelly@jden devops-platform-workshops-labs]$ git branch jefkel-201
[jkelly@jden devops-platform-workshops-labs]$ git checkout jefkel-201
Switched to branch 'jefkel-201'
[jkelly@jden devops-platform-workshops-labs]$ git push --set-upstream origin jefkel-201
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'jefkel-201' on GitHub by visiting:
remote:     https://github.com/jefkel/devops-platform-workshops-labs/pull/new/jefkel-201
remote:
To github.com:jefkel/devops-platform-workshops-labs.git
 * [new branch]      jefkel-201 -> jefkel-201
Branch 'jefkel-201' set up to track remote branch 'jefkel-201' from 'origin'.
[jkelly@jden devops-platform-workshops-labs]$ 

```

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.

A screenshot of a GitHub forked repository page. At the top, there's a navigation bar with links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the navigation, the repository name 'jefkel / devops-platform-workshops-labs' is shown, along with a 'Watch' button (0), a 'Star' button (0), and a 'Fork' button (1). The main content area shows tabs for 'Code', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. Below these tabs, there are buttons for 'Overview', 'Yours' (which is selected), 'Active', 'Stale', and 'All branches'. To the right of these buttons is a search bar labeled 'Search branches...'. Under the 'Yours' tab, a section titled 'Your branches' lists a single branch: 'jefkel-201' (Updated 4 minutes ago by jefkel). To the right of this branch listing is a progress bar at 0% and a 'New pull request' button. At the bottom of the page, there's a footer with links for 'Contact GitHub', 'Pricing', 'API', 'Training', 'Blog', and 'About'.

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

IMPORTANT Change the base repository to be {admin}/devops-platform-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)

A screenshot of the BCDevOps repository's 'Open a pull request' interface. The top navigation bar is identical to the previous screenshot. The main area shows the repository 'BCDevOps / devops-platform-workshops-labs' with a 'Watch' button (4), a 'Star' button (0), and a 'Fork' button (1). Below the repository name, there are tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', and 'Insights'. The 'Pull requests' tab is selected. The main content area is titled 'Open a pull request' and contains instructions: 'Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.' Below these instructions is a form for setting up the pull request. It includes dropdowns for 'base repository' (set to 'BCDevOps/devops-platform-labs'), 'base' (set to 'master'), 'head repository' (set to 'jefkel/devops-platform-workshops-labs'), and 'compare' (set to 'jefkel-201'). There are also sections for 'Reviewers' (No reviews) and 'Assignees' (No one assigned). At the bottom of the form is a 'Leave a comment' input field. A blue box highlights this input field, which contains the following text: 'you can cut and paste the following into the comment section to start a Lab checklist. Add more as you go!' Below this, another blue box highlights the text 'My Lab Checklist:' followed by a bulleted list of tasks:

- [] 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.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub pull request creation interface. At the top, it says "base: master" and "compare: jefkel-201". A green checkmark indicates "Able to merge. These branches can be automatically merged." Below this, there's a text area titled "Jefkel 201" with tabs for "Write" and "Preview". The "Write" tab contains a "My Lab Checklist:" section with a bulleted list of tasks related to GitHub PRs and Jenkins. Below the checklist is a file upload area. To the right, there are sections for "Reviewers" (No reviews), "Assignees" (No one—assign yourself), "Labels" (None yet), "Projects" (None yet), and "Milestone" (No milestone). At the bottom left, it shows "2 commits" and "1 file changed". On the right, it shows "1 contributor" and the commit hash "08b3c77". A tooltip is overlaid on the "Draft pull request" button, which says "Create pull request Open a pull request that is ready for review" and "Create draft pull request Cannot be merged until marked 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!

The screenshot shows a GitHub pull request page for the repository `jefkel / devops-platform-workshops-labs`. The repository is a fork from `BCDevOps/devops-platform-workshops-labs`. The pull request is titled "Jefkel 201 #1" and is currently in draft mode, merging two commits from the `jefkel-201` branch into the `master` branch. The pull request has 1 approval and 0 comments.

Key details from the pull request page:

- Owner:** jefkel
- Reviewers:** No reviews
- Assignees:** jefkel
- Labels:** None yet
- Projects:** None yet
- Milestone:** No milestone
- Notifications:** None yet

The pull request interface includes tabs for Code, Pull requests (1), Projects (0), Wiki, Insights, and Settings. The main content area displays the commit history, showing a comment from jefkel about a checklist, several commits added by jefkel, and a self-assigned status.

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 apply -f -
```

- Monitor the status of the object creation

```
oc status
```

- From the GUI, the template can also be viewed

Name	Created	Labels
postgresql-ephemeral	12 minutes ago	none

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
```

- Navigate 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,\nserviceAccounts.server.create=false,\nrbac.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\nkubernetes_sd_configs:\n- role: pod\n  namespaces:\n    names:\n      - [ PROJECT NAME ]\n  relabel_configs:\n    - action: keep\n      regex: true\n      source_labels:\n        - __meta_kubernetes_pod_annotation_prometheus_io_scrape\n    - action: replace\n      regex: (.+)\n      source_labels:\n        - __meta_kubernetes_pod_annotation_prometheus_io_path\n      target_label: __metrics_path__\n    - action: replace\n      regex: ([^:]+)(?:\d+)?;(\d+)\n      replacement: $1:$2\n      source_labels:\n        - __address__\n        - __meta_kubernetes_pod_annotation_prometheus_io_port\n      target_label: __address__\n    - action: labelmap\n      regex: __meta_kubernetes_pod_label_(.+)\n    - action: replace\n      source_labels:\n        - __meta_kubernetes_namespace\n      target_label: kubernetes_namespace\n    - action: replace\n      source_labels:\n        - __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

The screenshot shows two main sections of the Prometheus UI:

Service Discovery

This section lists active targets:

- kubernetes-pods (0/2 active targets)
- prometheus (1/1 active targets)

For the **prometheus** target (selected), it shows discovered labels and target labels:

Discovered Labels	Target Labels
address: "localhost:9090"	instance:"localhost:9090"
metrics_path: "/metrics"	pod:"prometheus"
scheme: "http"	
job:"prometheus"	

Targets

This section lists targets by endpoint:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
kubernetes-pods (0/0 up) <small>(show less)</small>					
prometheus (1/1 up) <small>(show less)</small>	UP	instance:"localhost:9090" job:"prometheus"	11.427s ago	6.21ms	
http://localhost:9090/metrics	UP				

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
```

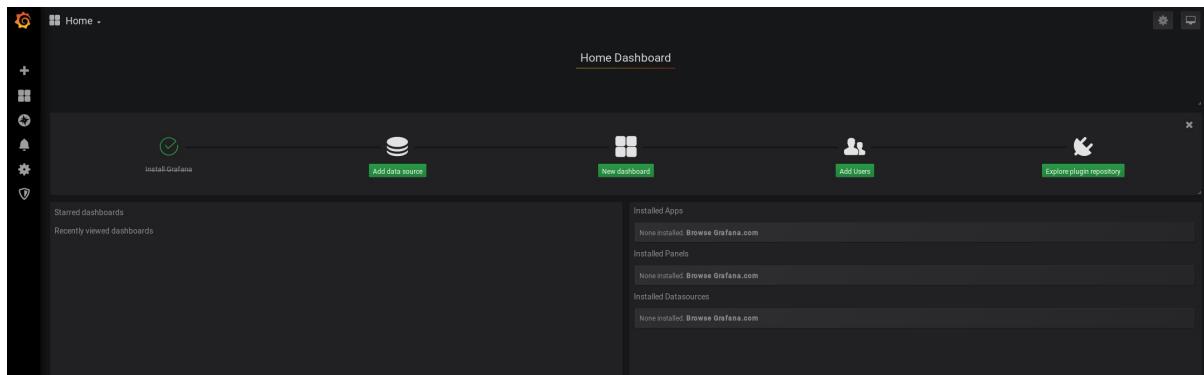
- Explore the output of the `new-app` command
- Explore the objects generated from the utility

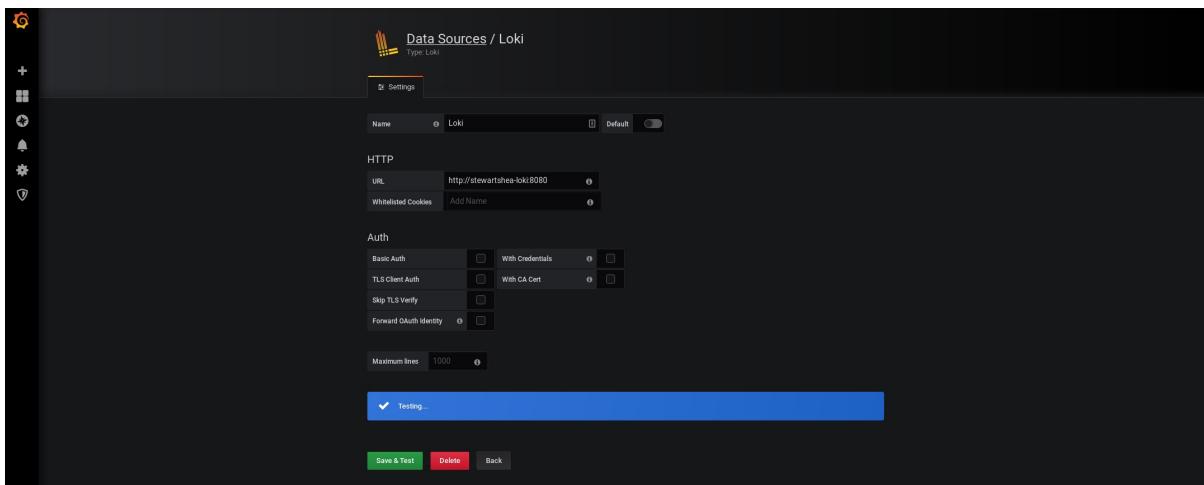
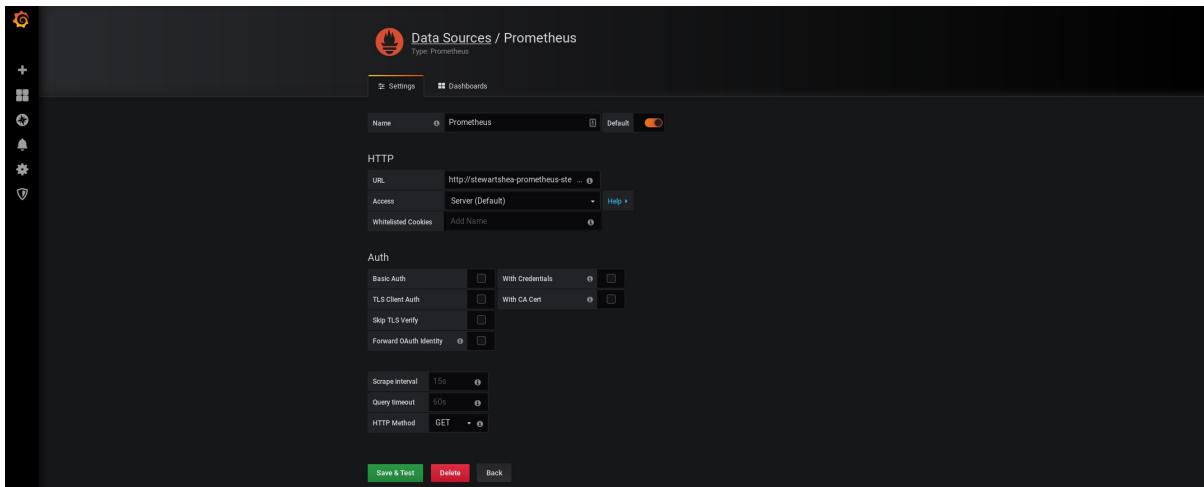
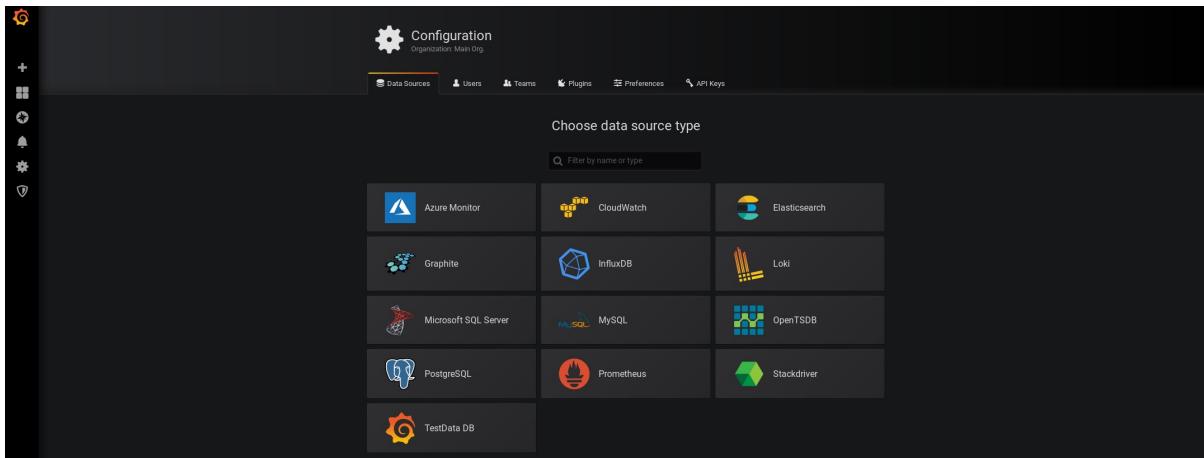
```
oc get all -l app=[username]-grafana
```

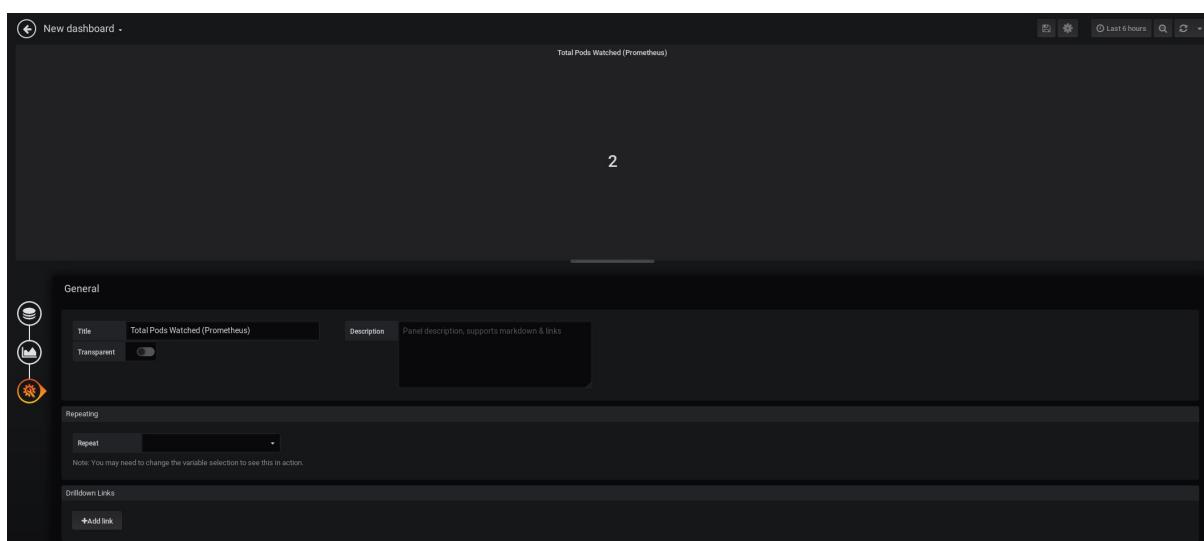
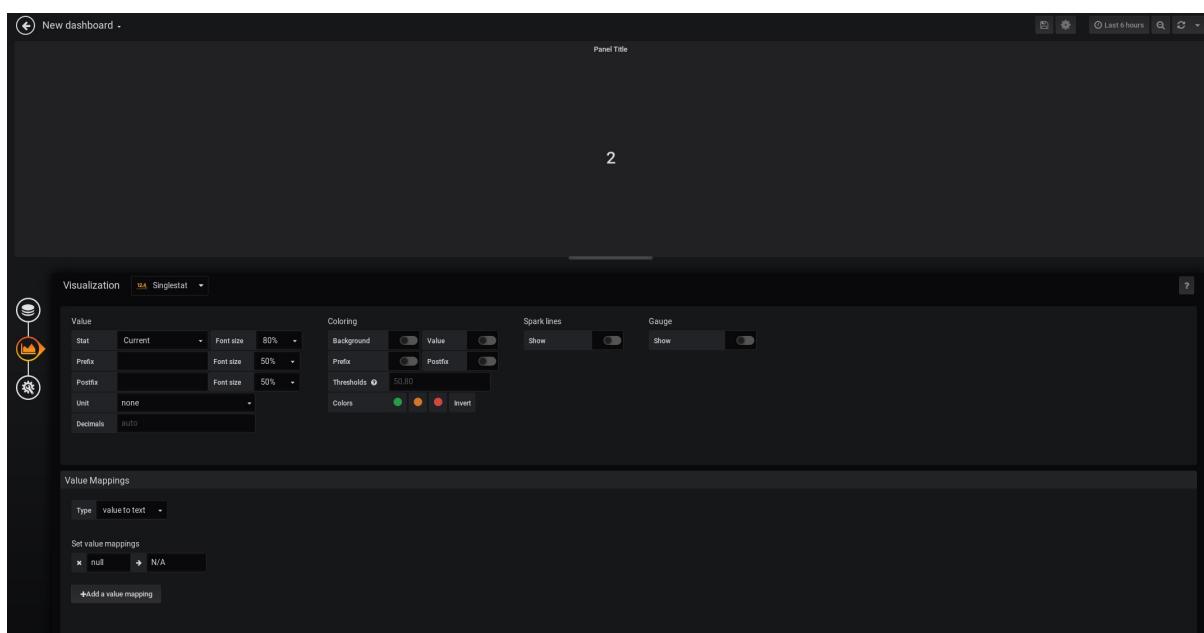
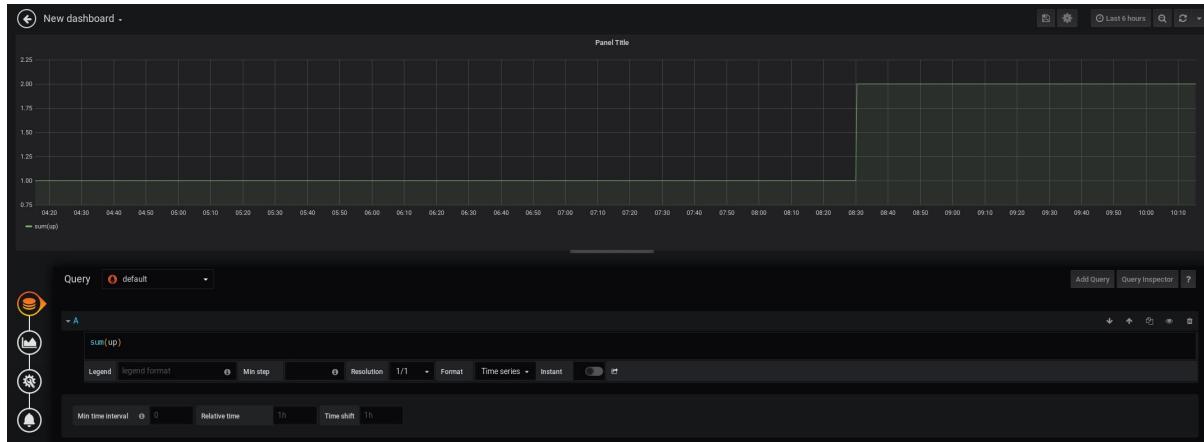
- Expose the Grafana service

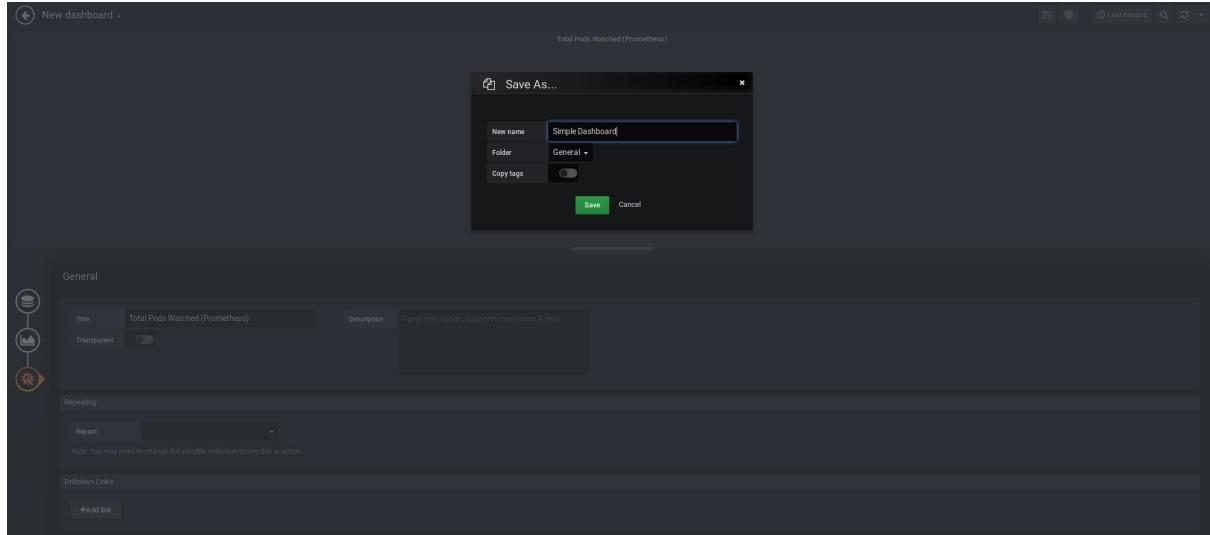
```
oc expose service [username]-grafana --name [username]-grafana
```

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

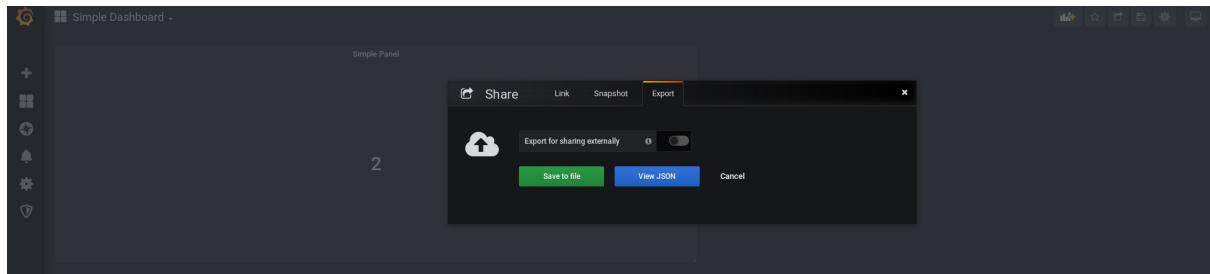








- Export the dashboard configuration for later use



- 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) ▾

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 MB
- Ports: 3000/TCP

Volumes

Add Storage | Add Config Files

Triggers

Manual (CLI): oc rollout latest dc/stewartshea-grafana -n 798

Learn More ⓘ

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) ▾

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 MB
- Ports: 3000/TCP

Volumes

Add Storage | Add Config Files

Triggers

Manual (CLI): oc rollout latest dc/stewartshea-grafana -n 798

Learn More ⓘ

Change Of: Config

New Image For: 7982jz-husker-bsg-openshift201-may2019-dev/stewartshea-grafana:6.2.0

Show Annotations

The screenshot shows the OpenShift Container Platform Application Console interface. On the left is a sidebar with navigation links: Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The main content area shows the path Deployments > stewartshea-grafana > Add Config Files. The title is "Add Config Files to stewartshea-grafana". It says "Add values from a config map or secret as volume. This will make the data available as files for deployment config stewartshea-grafana." There are fields for "Source" (with a dropdown menu "Select config map or secret" and buttons "Create Config Map" and "Create Secret"), "Mount Path" (example: "/data"), and a checkbox "Select specific keys and paths" with the note "Add only certain keys or use paths that are different than the key names." At the bottom are "Add" and "Cancel" buttons.

- 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
  
```

The screenshot shows the 'Create Config Map' interface in the OpenShift Application Console. The 'Name' field is set to 'stewartshea-grafana-dashboard-providers'. The 'Key' field is set to 'dashboards.yml'. The 'Value' field contains the following YAML code:

```

10  # <string> folder UID. will be automatically generated if not specified
11  folderId: null
12  # <string> required provider type. Required
13  type: file
14  # <bool> disable dashboard deletion
15  deleteDashboards: false
16  # <bool> enable dashboard editing
17  editable: true
18  # <int> how often Grafana will scan for changed dashboards
19  updateIntervalSeconds: 10
20  options:
21    # <string> required path to dashboard files on disk. Required
22  path: /var/lib/grafana/dashboards

```

At the bottom, there are 'Remove Item' and 'Add Item' buttons, followed by 'Create' and 'Cancel' buttons.

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

The screenshot shows the 'Add Config Files' interface for deployment 'stewartshea-grafana'. The 'Source' is set to 'stewartshea-grafana-dashboard-providers - Config Map'. The 'Mount Path' is set to '/etc/grafana/provisioning/dashboards/'. The 'Select specific keys and paths' checkbox is checked. At the bottom, there are 'Add' and 'Cancel' buttons.

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

```

apiVersion: v1
data:
  dashboards.yaml: |
    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
kind: ConfigMap
metadata:
  creationTimestamp: '2019-05-26T18:58:37Z'
  labels:
    app: stewartshea-grafana
  name: stewartshea-grafana-dashboard-providers
  namespace: 7982j-z-husker-bsg-openshift201-may2019-dev
  resourceVersion: '793713166'
  selfLink: >
    /api/v1/namespaces/7982j-z-husker-bsg-openshift201-may2019-dev/configmaps/stewartshea-grafana-dashboard-providers
  uid: 44de30d9-7fe8-11e9-8dc7-0050568348cc

```

- 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

```

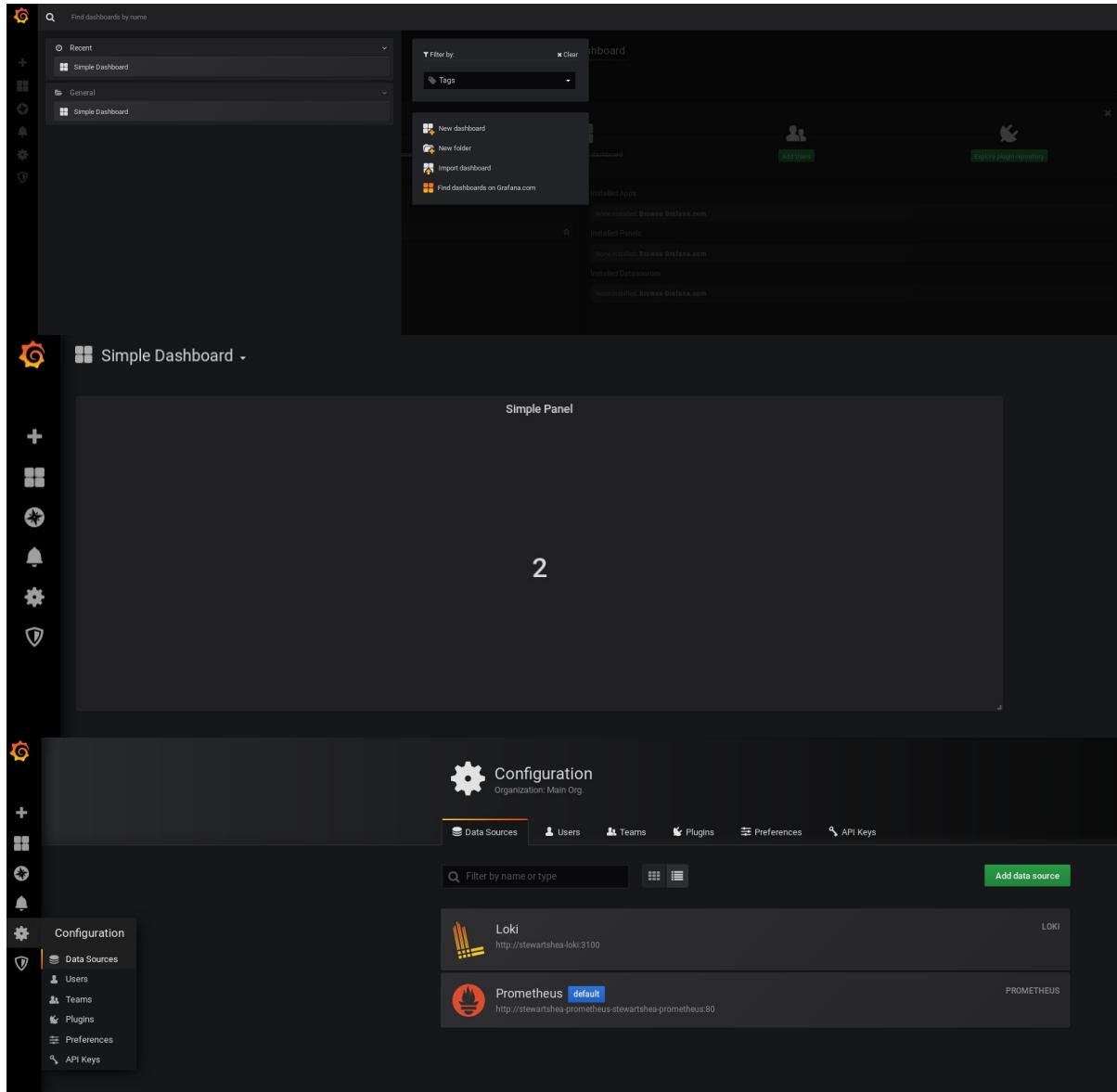
- **dashboards** mount path: `/var/lib/grafana/dashboards/`
- **dashboard** key: `simple_dashboard.json`
- **dashboard content:** json content from the dashboard export

Note Remember to add the appropriate labels to all configMaps

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

The screenshot shows the OpenShift Container Platform Application Console. The left sidebar has navigation links: Overview, Applications (selected), Builds, Resources, Storage, Monitoring, and Catalog. The main content area shows the 'Deployments' section for the application 'stewartshea-grafana'. The 'Configuration' tab is selected. On the right, there's a 'Details' panel with deployment configuration details like Replicas (1), Strategy (Rolling), and Timeout (600 sec). Below it is a 'Template' panel with a note about health checks. At the bottom, there's a 'Volumes' section listing three volumes: 'volume-bfku7', 'volume-ttg2w', and 'volume-y3co5', each associated with a 'Config Map'.

- 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 manifest
- Edit the manifest to make it more "generic"
 - Remove any references to:
 - lastApplied 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 succeed 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_NAME}.
  For more information about using this template, including OpenShift considerations, 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. ConfigMaps 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-workshops
    openshift.io/long-description: This template provides a sample Grafana configuration 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  name:\n    'default'\n    # <int> org id. will default to orgId 1 if not specified\n    orgId:\n      1\n      # <string, required> name of the dashboard folder. Required\n      folder:\n        ''\n        # <string> folder UID. will be automatically generated if not specified\n\n    \\\n    folderUid: ''\n    # <string, required> provider type. Required\n    type: file\n\n    \\\n    # <bool> disable dashboard deletion\n    disableDeletion: false\n    # <bool>\n    enable dashboard editing\n    editable: true\n    # <int> how often Grafana will\n    scan for changed dashboards\n    updateIntervalSeconds: 10\n    \\\n    options:\n      # <string, required> path to dashboard files on disk. Required\n      path: /var/\n      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: loki\n  orgId: 1\n  access: proxy\n  url: http://${LOKI_SERVICE_NAME}:3100\n  json\nData:\n    \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 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 -
```

Committing 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 `.pipeline` 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`

The screenshot shows the OpenShift web console interface. The top navigation bar includes 'OPENSHIFT CONTAINER PLATFORM', 'Application Catalog', and user information. The left sidebar has links for Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The main content area is titled 'Pipelines' with a 'Recent Runs' section. A single pipeline run is listed: 'devops-platform-workshops-labs' (created a few seconds ago). The pipeline has a single step named 'Build Stage' which is currently running. A progress bar shows the step is at 45%. Below the pipeline run, there are links for 'View Pipeline Runs' and 'Edit Pipeline'. The status bar at the bottom right shows 'Average Duration: 8s'.

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

The screenshot shows the OpenShift web console interface, specifically the 'Configuration' tab for a pipeline. The left sidebar is identical to the previous screenshot. The main content area shows the pipeline details for 'build devops-platform-workshops-labs'. Under the 'Details' section, it shows the Jenkins Pipeline configuration with source repository 'stewartshea/devops-platform-workshops-labs'. The 'Triggers' section is expanded, showing the configuration for the webhook. It includes fields for 'Config Change For', 'Generic Webhook URL' (set to 'https://console.pathfinder.gov.bc.ca:8443/apis/...'), 'GitHub Webhook URL' (set to 'https://console.pathfinder.gov.bc.ca:8443/apis/...'), and 'Manual (CLI)' (set to 'oc start-build devops-platform-workshops-labs')).

The screenshot shows the Jenkins Settings interface with the 'Webhooks' section selected. On the left sidebar, 'Webhooks' is highlighted. The main area displays the 'Add webhook' configuration page. It includes fields for 'Payload URL' (set to <https://console.pathfinder.gov.bc.ca:8443/apis/build.openshift.io/v1/>), 'Content type' (set to 'application/json'), and a 'Secret' field. Under 'SSL verification', there are two options: 'Enable SSL verification' (selected) and 'Disable (not recommended)'. Below that, under 'Which events would you like to trigger this webhook?', there are three radio button options: 'Just the push event.' (selected), 'Send me everything.', and 'Let me select individual events.'. A checked checkbox labeled 'Active' indicates that event details will be delivered when triggered. At the bottom is a green 'Add webhook' button.

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

The screenshot shows the OpenShift 201 dashboard. The left sidebar lists various project categories: Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The 'Builds' category is expanded, showing a list of pipelines. One pipeline, 'devops-platform-workshops-labs', is selected and shown in detail. The pipeline history tab indicates that 'Build #2 is complete' and was started 'a few seconds ago'. The build stage status shows a single stage named 'Build Stage' with a progress bar at 23%. There is also a table showing two builds, with the second build having a status of 'Build Stage' and a progress bar at 45%.

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

The screenshot shows the Jenkins interface for a completed build of the 'devops-platform-workshops-labs' pipeline. The build stage is labeled 'Build Stage' and is marked as successful. The log output at the bottom shows the command 'sh "oc get projects"' being executed.

```

sh "oc get projects"

```

- Jenkins also provides a [Pipeline Syntax](#) tool with a Snippet Generator to assist in generating pipeline code

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface. On the left, there's a sidebar with various links like Up, Status, Changes, Build Now, Delete Pipeline, Configure, Move, Full Stage View, Open Blue Ocean, Rename, and Pipeline Syntax. The main area has a title "Pipeline 7982jz-husker-bsg-openshift201-may20" and a subtitle "Stage View". Below that is a "Build Stage" card showing "Average stage times: 3s". The main content area is titled "Sample Step: openshiftDeploy: Trigger OpenShift Deployment". It includes fields for "Cluster API URL (only clusters at v3.11 or earlier are supported)", "Authorization Token", "Project", and "DeploymentConfig name". There are three warning messages about API endpoints, tokens, and namespaces. At the bottom, there's a "Test Connection" button and an "Advanced..." link. A "Generate Pipeline Script" button is at the very bottom.

```

#!/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

```

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.yaml
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'

```

The screenshot shows the Jenkins OpenShift pipeline interface. At the top, there's a header with the pipeline name '7982jz-husker-bsg-openshift201-may2019-tools / 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs'. Below the header, it shows 'Branch: -' and 'Commit: -'. The main area displays a pipeline step titled 'Deploy Loki to Dev'. This step has three stages: 'Start', 'Dev' (which is highlighted with a blue circle), and 'End'. Underneath this, there's a detailed log for the 'Deploy Loki to Dev' step, which includes a list of tasks and their execution times.

Task	Time
> General SCM	2s
> oc process -f openshift201/templates/template_loki_sample.yaml -p LOKI_SERVICE_NAME=loki-9 oc apply -f - -n 7982jz-husker-bsg-openshift201-may2019-dev	1s
> echo Wait for service to be up	<1s
Verify OpenShift Service	51s

```

Deploy Loki to Dev - 59s
[✓] > General SCM
[✓] > oc process -f openshift201/templates/template_loki_sample.yaml -p LOKI_SERVICE_NAME=loki-9 | oc apply -f - -n 7982jz-husker-bsg-openshift201-may2019-dev
-> Shell Script
[✓] > echo Wait for service to be up
-> Shell Script
[✓] Deploy Loki to Dev
[✓] Start
[✓] Dev
[✓] End

```

- 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.yaml -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_SERVICE}")
                        // this will wait until the desired replicas are available
                        dc.rollout().status()
                    }
                }
            }
        }
    }
}
```

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

Branch: - Commit: - 17s No changes Started by user husker-bsg

Description OpenShift Build 7982j-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-21 from https://github.com/stewartshea/devops-platform-workshops-labs

Deploy Loki to Dev - 13s

- > General SCM
- > oc process -f openshift201/templates/template_loki_sample.yaml -p LOKI_SERVICE_NAME=loki | oc apply -f - -n 7982j-husker-bsg-openshift201-may2019-dev — Shell Script
- > echo Wait for service to be up — Shell Script
- > Print Message
- > Internal utility function for OpenShift DSL
- > Internal utility function for OpenShift DSL
- > /var/run/secrets/kubernetes.io/serviceaccount/token — Read file from workspace
- Internal utility function for OpenShift DSL

[...]

Waiting for deployment "loki" rollout to finish: 0 of 1 updated replicas are available...

- 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"

```

Branch: - Changes by stewart.shea
 Commit: - OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-23 from https://github.com/stewartshea/devops-platform-workshops-labs, commit f1429ce95590bcf6c116905b1f3ad3a39cc527d8



Deploy Loki to Dev - 17s

```

✓ > General SCM
✓ > oc process -f openshift201/templates/template_loki_sample.yaml -p LOKI_SERVICE_NAME=loki | oc apply -f -n 7982jz-husker-bsg-openshift201-may2019-dev -- Shell Script
✓ > echo Wait for service to be up — Shell Script
✓ > Print Message
✓ > Internal utility function for OpenShift DSL
✓ > Internal utility function for Openshift DSL
✓ > /var/run/secrets/kubernetes.io/serviceaccount/token — Read file from workspace
○ > Internal utility function for OpenShift DSL
  [rollout:status:deployment/loki] Waiting for deployment "loki" rollout to finish: 1 old replicas are pending termination...

```

- 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

Branch: - 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 9fd29e53acdb0cb8f23e3104ede5c3e085f1d1c2...



Wait for Validation - 17s

II > Wait for interactive input

Want to deploy to PROD?

HANG TIGHT! Abort

7982jz-husker-bsg-openshift201-may2019-tools / 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs < 24

Branch: - 50s Changes by stewart.shea
Commit: - OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-24 from https://github.com/stewart.shea/devops-platform-workshops-labs, commit 9fd29e53acdb0cb8523e3104ede5c3e085f1d1c1...

Description: OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-24 from https://github.com/stewart.shea/devops-platform-workshops-labs, commit 9fd29e53acdb0cb8523e3104ede5c3e085f1d1c1...

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

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

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

        echo "Waiting for builds to complete..."

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

}
}
```

- Create the buildConfig inside of the `tools` namespace

```
oc new-build https://github.com/[username]/devops-platform-workshops-labs.git#[username]-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 ('Build in Tools Namespace'){
        dir ('simple_pipeline') {
            checkout scm
            timeout (time: 600, unit: 'SECONDS'){
                openshift.withCluster() {
                    openshift.withProject("${TOOLS_NAMESPACE}") {
                        def blackboxSelector = openshift.selector("bc", "blackboxexporter")
                        try {
                            blackboxSelector.object()
                            builds = blackboxSelector.related( "builds" )
                        } catch (Throwable t) {
                            nb = openshift.newBuild( "https://github.com/[username]/devops-platform-workshops-labs.git#[username]-201", "--context-dir=blackbox_exporter", "--name=blackboxexporter" )

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

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

```

```

        }
    }
}

}

}

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}/blackboxexporter:dev", "--name=blackboxexporter").narrow('svc').expose()
            }

            // Print out information about the objects created 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"
        }
    )
}
```

The screenshot shows a Jenkins pipeline run titled "7982jz-husker-bsg-openshift201-may2019-tools / 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs". The pipeline has three parallel steps: "grafana", "loki", and "prometheus". Each step consists of a "Deploy" phase with a green status indicator. The pipeline summary at the bottom shows the status of each step.

Step	Status	Duration
grafana	Success	<1s
loki	Success	<1s
prometheus	Success	<1s

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

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

The screenshot shows two Jenkins pipeline runs. Both runs have the same configuration:

- Branch:** -
- Commit:** -
- Changes by stewart.shea**
- OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-30** from <https://github.com/stewart.shea/devops-platform-workshops-labs>, commit 722454d457bbfb9b6a7252ead7a6fd69fe3479

Run 1 (Top):

- Description:** OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-30 from <https://github.com/stewart.shea/devops-platform-workshops-labs>, commit 722454d457bbfb9b6a7252ead7a6fd69fe3479
- Pipeline Status:** Wait for Validation - 16s
- Wait for Validation Step:** Wait for interactive input (status: HANG TIGHT!)
- Deployment Graph:** Shows three parallel tasks: grafana, loki, and prometheus. The grafana task has a dependency on the loki task, which in turn has a dependency on the prometheus task. All three tasks are marked as completed (green checkmarks).

Run 2 (Bottom):

- Description:** OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-30 from <https://github.com/stewart.shea/devops-platform-workshops-labs>, commit 722454d457bbfb9b6a7252ead7a6fd69fe3479
- Pipeline Status:** Deploy Prod Environment / grafana - 8s
- Deploy Prod Environment Step:** Internal utility function for OpenShift DSL (status: 4s)
- Deployment Graph:** Shows the same parallel tasks as Run 1, but the final "Deploy Prod Environment" step is shown as a separate step after the validation step. The grafana task is now marked as pending (blue circle).

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

```

1 level=warn ts=2019-05-28T01:28:05.763380884Z caller=filetargetmanager.go:101 msg="WARNING!!! entry.parser config is deprecated, please change to pipeline_stages"
2 level=info ts=2019-05-28T03:28:05.764540423Z caller=server.go:120 http=:1::1:9080 grpc=:1::1:91637 msg="server listening on addresses"
3 level=info ts=2019-05-28T03:28:05.76493852Z caller=main.go:49 msg="Starting Prontal* version=(version:master-39bbd73, branch:master, revision=39bbd73)"
4 level=info ts=2019-05-28T03:28:05.76493852Z caller=main.go:243 msg="Adding target" key="(jobName:grafana*)"
5 level=info ts=2019-05-28T03:28:05.76493852Z caller=tailer.go:68 msg="start tailing file" path="/logs/grafana.log"
6 2019/05/28 03:28:10 Seeked /logs/grafana.log + 6(Offset:0 Whence:0)

```


This datasource was added by config and cannot be modified using the UI. Please contact your server admin to update this datasource.

Name	Loki	<input type="checkbox"/> Default
HTTP		
URL	http://loki:3100	<input type="button"/>
Whitelisted Cookies	Add Name	<input type="button"/>
Auth		
Basic Auth	<input type="checkbox"/> With Credentials	
TLS Client Auth	<input type="checkbox"/> With CA Cert	
Skip TLS Verify	<input type="checkbox"/>	
Forward OAuth Identity	<input type="checkbox"/>	
Maximum lines 1000		

✓ Data source connected and labels found.

Test **Delete** **Back**

Common labels: /logs/grafana.log grafana Limit: 1000 (269 returned)

```

2019-05-28T14:18:37.072837744Z 2019-05-28 07:18:37 t=2019-05-28T14:18:37>@0000 lv=info msg="Request Completed" logger=context userId=0 orgId=0 uname= method=GET path=/status?addr=24.69.149.118 time_ms=0 size=29 referer= http://grafana-62-ljszt:3000/
2019-05-28T14:18:37.072865406Z 2019-05-28 07:18:37 t=2019-05-28T14:18:37>@0000 lv=info msg="Request Completed" logger=context userId=0 orgId=0 uname= method=GET path=/datasources/edit/2/ status=302 remote_addr=24.69.149.118 time_ms=1 size=24 referer= http://grafana-62-ljszt:3000/
2019-05-28T14:18:37.072645932Z 2019-05-28 07:18:37 t=2019-05-28T14:18:37>@0000 lv=error msg="Failed to look up user based on cookie" logger=context error="user token not found"
2019-05-28T14:18:28.465253190Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="HTTP Server Listen" logger=http.server address=0.0.0.0:3000 protocol=http suburi= socket=
2019-05-28T14:18:28.465224832Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing Stream Manager"
2019-05-28T14:18:28.465157908Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Inserting datasource from configuration" logger=provisioning.datasources name=Loki
2019-05-28T14:18:28.465155831Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Inserting datasource from configuration" logger=provisioning.datasources name=Prometheus
2019-05-28T14:18:28.465187832Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing provisioningServiceImpl" logger=server
2019-05-28T14:18:28.465084585Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing NotificationService" logger=server
2019-05-28T14:18:28.465083652Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing CleanUpService" logger=server
2019-05-28T14:18:28.464995962Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing UserAuthService" logger=server
2019-05-28T14:18:28.464995962Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing SessionService" logger=server
2019-05-28T14:18:28.464923748Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing TracingService" logger=server
2019-05-28T14:18:28.464989675Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing ServerLockService" logger=server
2019-05-28T14:18:28.464975602Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing SearchService" logger=server
2019-05-28T14:18:28.464973212Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing HooksService" logger=server
2019-05-28T14:18:28.464950877Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing DatasourceCacheService" logger=server
2019-05-28T14:18:28.464924921Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing AlertingService" logger=server
2019-05-28T14:18:28.464921753Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Initializing RenderingService" logger=server
2019-05-28T14:18:28.464961577Z 2019-05-28 07:18:28 t=2019-05-28T14:18:28>@0000 lv=info msg="Starting plugin search" logger=plugins

```

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/pa
tronii/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

Stateful Sets > patroni

Actions	
patroni	
app	patroni
app.kubernetes.io/component	database
app.kubernetes.io/instance	patroni
More labels...	
Details	Environment
Metrics	Events
Status:	Active
Replicas:	3 replicas
	
Template	
-	

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)
```

The screenshot shows the deployment configuration for the 'jefkel-grafana' application. At the top, there are tabs for 'Deployments' and 'Actions'. Below that, the deployment name 'jefkel-grafana' is shown with a creation timestamp of 'created 3 minutes ago'. There are two tabs: 'app' and 'jefkel-grafana', with 'app' being active. Under the 'Environment' tab, the configuration for the 'Container jefkel-grafana' is listed. It includes five environment variables with their corresponding values and dropdown menus for secrets:

Name	Value
GF_DATABASE_TYPE	postgres
GF_DATABASE_HOST	patroni-master
GF_DATABASE_NAME	patroni - Secret
GF_DATABASE_USER	patroni - Secret
GF_DATABASE_PASSWORD	patroni - Secret

At the bottom of the configuration section, there are links for 'Add Value' and 'Add Value from Config Map or Secret'.

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.

```

17 2019-05-27 05:27:37.559 UTC [70] LOG: database system is ready to accept read only connection | Stop Following
18 2019-05-27 05:27:37.651 UTC [93] LOG: started streaming WAL from primary at 0/3000000 on timeline 1
19 localhost:5432 - accepting connections
20 2019-05-27 05:40:46.921 UTC [70] LOG: received fast shutdown request
21 2019-05-27 05:40:46.929 UTC [70] LOG: aborting any active transactions
22 2019-05-27 05:40:46.929 UTC [93] FATAL: terminating walreceiver process due to administrator command
23 2019-05-27 05:40:46.929 UTC [181] FATAL: terminating connection due to administrator command
24 2019-05-27 05:40:46.929 UTC [97] FATAL: terminating connection due to administrator command
25 2019-05-27 05:40:47.017 UTC [90] LOG: shutting down
26 2019-05-27 05:40:47.122 UTC [70] LOG: database system is shut down
27 2019-05-27 05:40:47.631 UTC [6747] LOG: listening on IPv4 address "0.0.0.0", port 5432
28 2019-05-27 05:40:47.652 UTC [6747] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
29 2019-05-27 05:40:47.927 UTC [6749] LOG: database system was shut down in recovery at 2019-05-27 05:40:47
UTC
30 2019-05-27 05:40:47.927 UTC [6749] LOG: entering standby mode
31 2019-05-27 05:40:48.024 UTC [6749] LOG: redo starts at 0/5226498
32 2019-05-27 05:40:48.024 UTC [6749] LOG: consistent recovery state reached at 0/5226578
33 2019-05-27 05:40:48.024 UTC [6749] LOG: invalid record length at 0/5226578: wanted 24, got 0
34 2019-05-27 05:40:48.115 UTC [6747] LOG: database system is ready to accept read only connections
35 localhost:5432 - accepting connections
36 2019-05-27 05:40:48.251 UTC [6753] LOG: started streaming WAL from primary at 0/5000000 on timeline 1
37 localhost:5432 - accepting connections

```

[Go to Top](#)

- 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 configuration 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

backup created a day ago

[Deploy](#) [Actions](#)

[template](#) [backup-deployment](#)

History Configuration [Environment](#) Events

Container backup

Name	Value	≡	X
BACKUP_DIR	/backups/	≡	X
DATABASE_SERVICE_NAME	patroni-master	≡	X
POSTGRESQL_DATABASE	Set to the key app-db-name in secret patroni	≡	X
POSTGRESQL_USER	Set to the key app-db-username in secret patroni	≡	X
POSTGRESQL_PASSWORD	Set to the key app-db-password in secret patroni	≡	X
SLEEP	1d	≡	X

- 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
```

