

## Starter Labs (Python)

### WORKSHOP MODULES

- Workshop Summary
- Environment Overview
- Using Homeroom
- Architecture Overview of the ParksMap Application
- Exploring the CLI and Web Console
- Deploying Your First Container Image
- Scaling and Self Healing
- Exposing Your Application to the Outside World
- Exploring OpenShift's Logging Capabilities
- Role-Based Access Control
- Remote Access to Your Application
- Deploying Python Code
- Adding a Database (MongoDB)
- Application Health
- Automate Build and Deployment with Pipelines
- Automation for Your Application on Code Changes**
- Further Resources
- Workshop Links

# Automation for Your Application on Code Changes

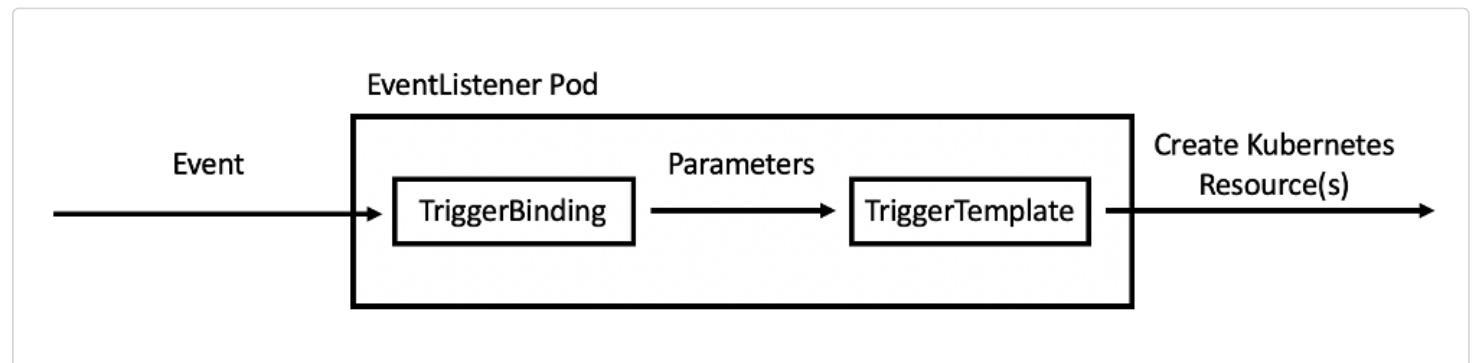
## Background: Web Hooks

Most Git repository servers support the concept of web hooks — calling to an external source via HTTP(S) when a change in the code repository happens. OpenShift provides an API endpoint that supports receiving hooks from remote systems in order to trigger builds. By pointing the code repository's hook at the OpenShift Pipelines resources, automated code/build/deploy pipelines can be achieved.

## Adding Triggers to your Pipeline

Tekton **Triggers** enable us to configure Pipelines to respond to external events (Git push events, pull requests etc) such as Web Hooks.

Adding triggering support requires the creation of a `TriggerTemplate`, `TriggerBinding`, and an `EventListener` in our project.



Let's see each component in detail:

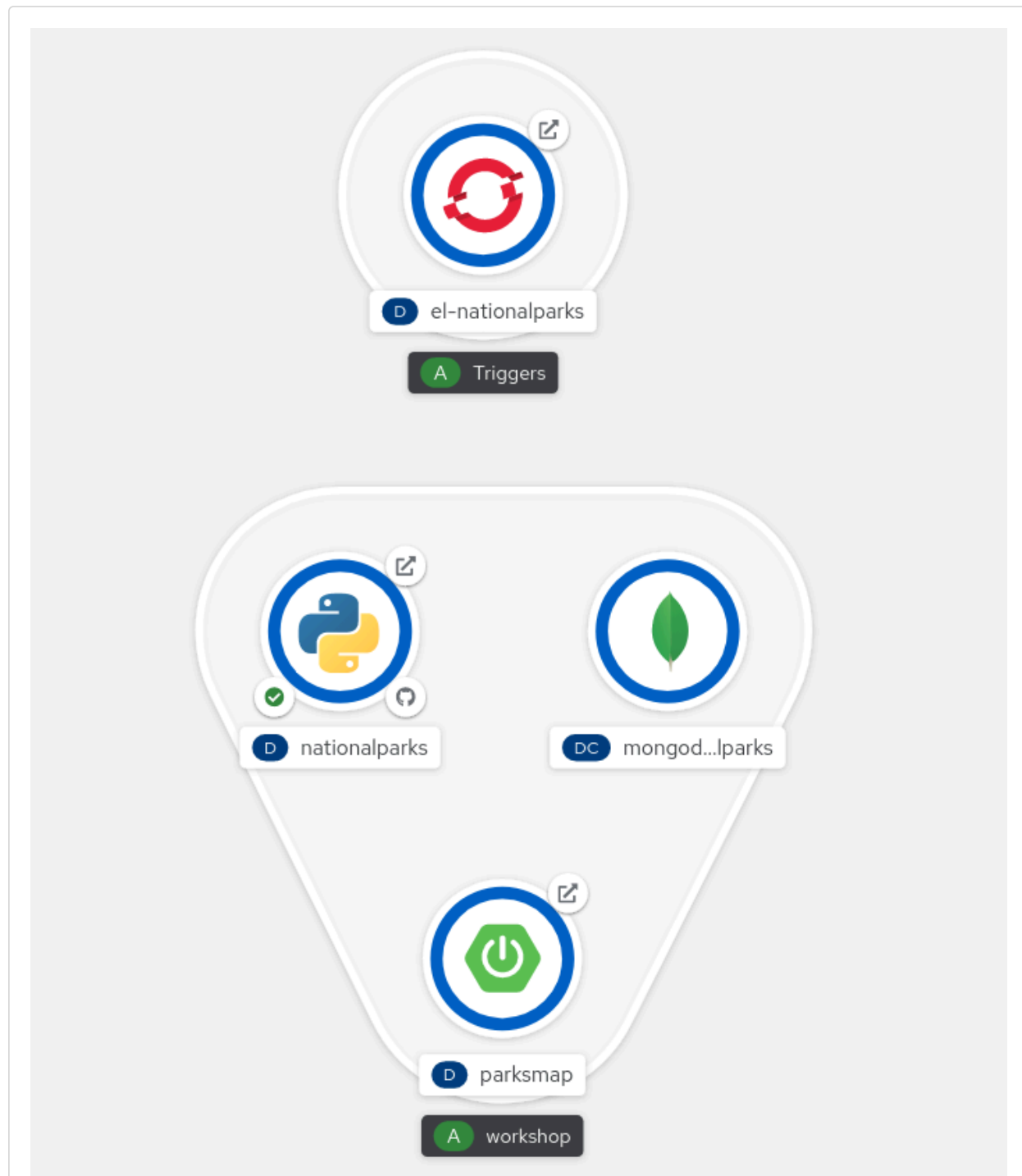
- **TriggerTemplate:** a trigger template is a template for newly created resources. It supports parameters to create specific `PipelineResources` and `PipelineRuns`.
- **TriggerBinding:** validates events and extracts payload fields
- **EventListener:** connects `TriggerBindings` and `TriggerTemplates` into an addressable endpoint (the event sink). It uses the extracted event parameters from each `TriggerBinding` (and any supplied static parameters) to create the resources specified in the corresponding `TriggerTemplate`. It also optionally allows an external service to pre-process the event payload via the `interceptor` field.

Now let's create them all together for our Pipeline:

```
oc create -f http://gogs-labs.apps.rosa-7s42b.rfax.p1.openshiftapps.com/user4/nationalparks-  
py/raw/master/pipelines/nationalparks-triggers-all.yaml -n user4
```

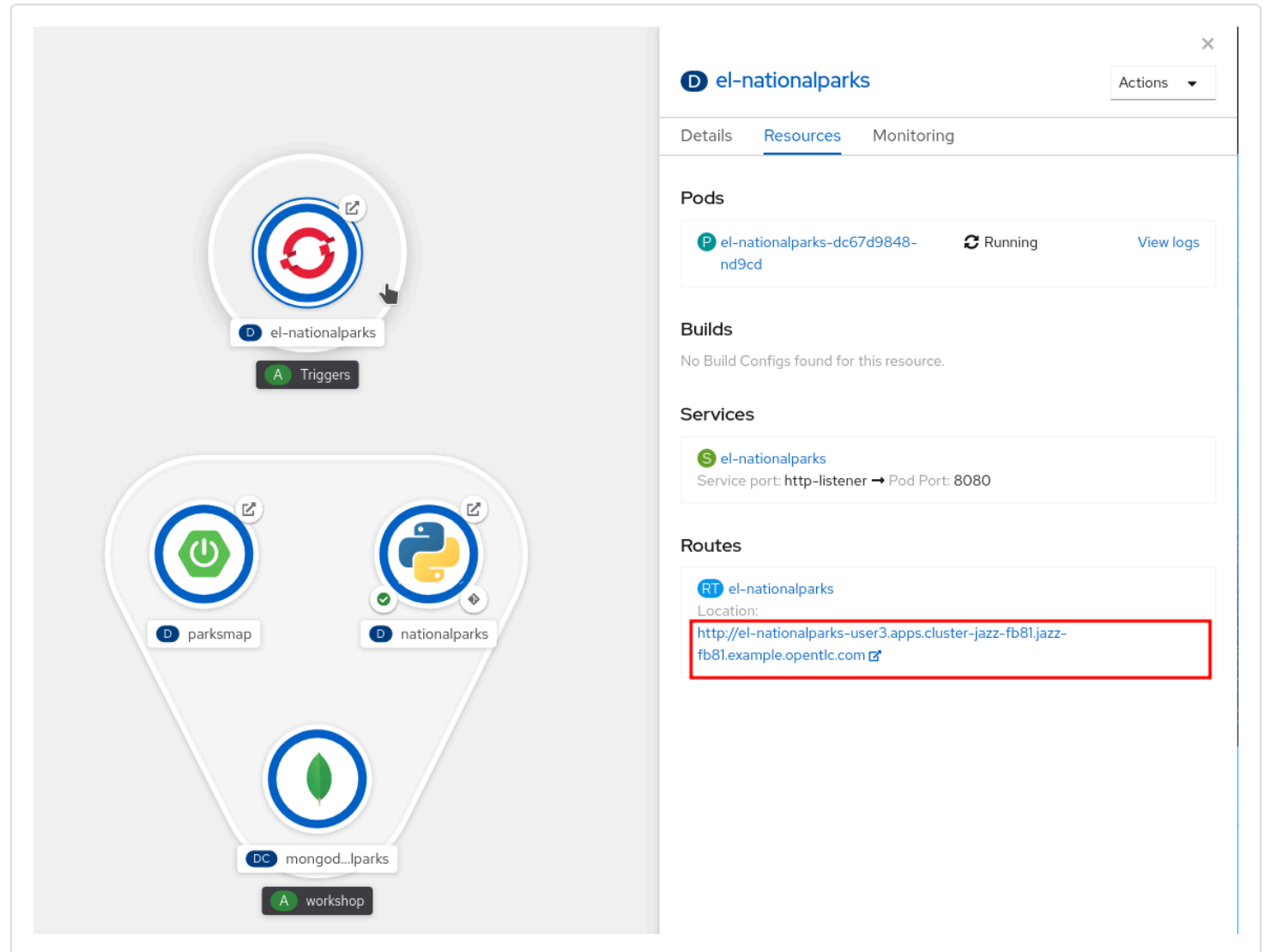
This will create a new Pod with a Route that we can use to setup our Webhook on Gogs to trigger the automatic start of the Pipeline.

From left side menu, click on **Topology** to verify if a new Deployment **el-nationalparks** for the `EventListener` has been created:



## Exercise: Configuring Gogs Web Hooks

Click on the Deployment, go into **Routes** section and copy the **el-nationalparks** Route URL.



The screenshot displays the OpenShift console interface. On the left, a diagram shows a deployment 'el-nationalparks' (represented by a circular icon with a refresh symbol) connected to a 'workshop' (represented by a green leaf icon). Below the deployment, there are two services: 'parksmapi' (represented by a green power icon) and 'nationalparks' (represented by a blue Python icon). The 'nationalparks' service is further connected to a 'mongodb...lparks' (represented by a blue leaf icon). On the right, the 'el-nationalparks' deployment details are shown. The 'Routes' section is highlighted, showing the 'el-nationalparks' route with the location: <http://el-nationalparks-user3.apps.cluster-jazz-fb81.jazz-fb81.example.opentlc.com>. The URL is enclosed in a red box.

**el-nationalparks** Actions

Details **Resources** Monitoring

**Pods**

Pod	Status	View logs
<b>P</b> el-nationalparks-dc67d9848-nd9cd	Running	<a href="#">View logs</a>

**Builds**

No Build Configs found for this resource.

**Services**

Service	Service port	Pod Port
<b>S</b> el-nationalparks	http-listener	8080

**Routes**

Route	Location
<b>RT</b> el-nationalparks	<a href="http://el-nationalparks-user3.apps.cluster-jazz-fb81.jazz-fb81.example.opentlc.com">http://el-nationalparks-user3.apps.cluster-jazz-fb81.jazz-fb81.example.opentlc.com</a>

Once you have the URL copied to your clipboard, navigate to the code repository that you have on Gogs:

## Gogs Repository

Your Gogs credentials are:

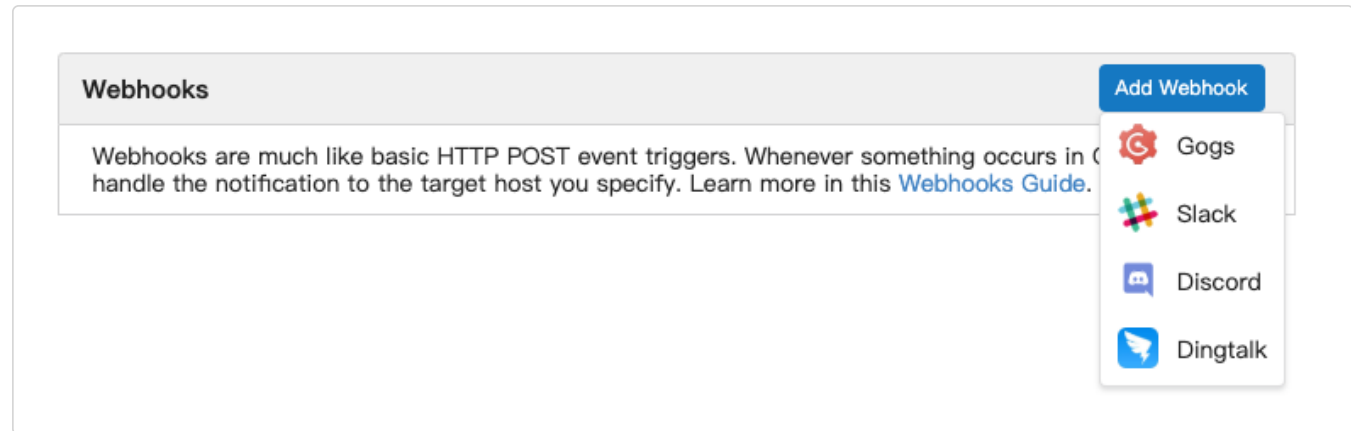
```
username: user4  
password: gogs
```

Click the Settings link on the top right of the screen:

The screenshot shows the Gogs repository settings page. At the top, there is a navigation bar with links for Files, Issues (0), Pull Requests (0), Wiki, and a highlighted Settings link. On the left, a sidebar contains links for Settings, Options, Collaboration, Branches, Webhooks (highlighted), and Deploy Keys. The main content area is titled 'Basic Settings' and contains the following fields:

- Repository Name \*: nationalparks
- Description: (empty text area)
- Official Site: (empty text area)
- Visibility: ☐ This repository is Private
- Update Settings button

Click on **Webhooks**, then the **Add Webhook** button, and finally select **Gogs**.



In the next screen, paste your link into the "Payload URL" field. You can leave the secret token field blank — the secret is already in the URL and does not need to be in the payload.

Change the `Content Type` to `application/json`.

Finally, click on **Add Webhook**.

Files

Issues 0

Pull Requests 0

Wiki

Settings

Settings

Options

Collaboration

Branches

Webhooks

Deploy Keys

Add Webhook

Gogs will send a POST request to the URL you specify, along with details regarding the event that occurred. You can also specify what kind of data format you'd like to get upon triggering the hook (JSON, x-www-form-urlencoded, XML, etc). More information can be found in our [Webhooks Guide](#).

Payload URL \*

http://el-nationalparks-user1.apps.cluster-982d.982d.example.opentlc.com/

Content Type

application/json

Secret

Secret will be sent as SHA256 HMAC hex digest of payload via X-Gogs-Signature header.

When should this webhook be triggered?

☒ Just the push event.

☐ I need **everything**.

☐ Let me choose what I need.

☒ Active

Details regarding the event which triggered the hook will be delivered as well.

Add Webhook

Boom! From now on, every time you commit new source code to your Gogs repository, a new build and deploy will occur inside of OpenShift. Let's try this out.

## Exercise: Using Gogs Web Hooks

Click the **Files** tab in Gogs. This is Gogs's repository view.






Make sure that the drop-down menu at the upper right is set for the `master` branch. Navigate to the following path:


/






Then click on the `wsgi.py` file.

Once you have the file on the screen, click the edit button in the top right hand corner as shown here:



 Files  Issues 0  Pull Requests 0  Wiki  Settings

 Branch: master ▾ nationalparks / wsgi.py

wsgi.py 3.2 KB     

```
1 from __future__ import print_function
2
3 import os
4 import json
5 import re
6
7 from flask import Flask, request
8 from flask_restful import Resource, Api
9
10 from pymongo import MongoClient, GEOS2D
11
12 DB_URI = os.environ.get('DB_URI')
13
14 DB_HOST = os.environ.get('DB_HOST', 'mongodb')
15
16 if os.environ.get('uri'):
17     print(os.environ.get('uri'))
18     match = re.match("mongodb?:\./\./([^\./]*):?(\d*)?", os.environ.get('uri'))
19
20     if match:
21         print(match.group())
22         DB_HOST = match.group(1)
23
24 DB_NAME = os.environ.get('DB_NAME', 'mongodb')
25
26 DB_USERNAME = os.environ.get('DB_USERNAME', 'mongodb')
27 DB_PASSWORD = os.environ.get('DB_PASSWORD', 'mongodb')
28
29 if not DB_URI:
30     DB_URI = 'mongodb://s:s@s:27017/s' % (DB_USERNAME, DB_PASSWORD,
31     DB_HOST, DB_NAME)
32
33 DATASET_FILE = 'nationalparks.json'
34
35 application = Flask(__name__)
36
37 api = Api(application)
38
39 class HealthCheck(Resource):
40     def get(self):
41         return 'OK'
42
43 api.add_resource(HealthCheck, '/ws/healthz/')
44
```

```
45 class Info(Resource):
46     description = {
47         'id': 'nationalparks-py',
48         'displayName': 'National Parks (PY)',
```

Change line number 50:

```
'displayName': 'National Parks (PY)',
```

To

```
'displayName': 'Amazing National Parks (PY)',
```

Click on Commit changes at the bottom of the screen. Feel free to enter a commit message.

Once you have committed your changes, a new **PipelineRun** should almost instantaneously be triggered in OpenShift. Click **Pipeline** in the left navigation menu then `nationalparks-pipeline`. You should see a new one running:

Red Hat OpenShift Container Platform

Project: user1

### Pipelines

Create Pipeline

Filter by name...

0 Complete 1 Running 0 Failed 0 Cancelled Select all filters 1 Item

Name	Namespace	Last Run	Task Status	Last Run Status	Last Run Time
nationalparks-pipeline	user1	nationalparks-deploy-zrfjn	Running	Running	-

or run the following command to verify:

```
oc get PipelineRun
```

Once the build and deploy has finished, verify your new image was automatically deployed by viewing the application in your browser:

[National Parks Info Page](#)

You should now see the new name you have set in the JSON string returned.

To see this in the map's legend itself, you will need to scale down your parksmap to 0, then back up to 1 to force the app to refresh its cache.

Continue