

Running an OpenShift Environment on Your Workstation

June 2018 | Leon Levy – Solution Architect
[`<llevy@kovarus.com>`](mailto:<llevy@kovarus.com>)

AGENDA

- OpenShift Overview
- MiniShift Overview and Install
- Logging Into OpenShift
- Deploy and Manage a Simple Application
- Java Microservice Application Demo
- Additional Topics for Next Time

OpenShift Container Platform

“vCenter for Containers?”

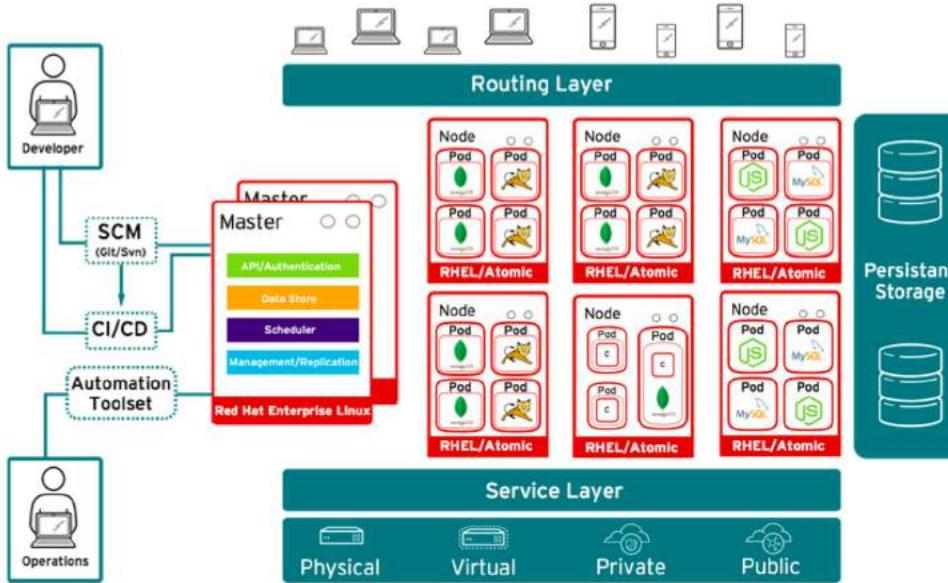
Unite developers and IT operations on a single platform to build, deploy, and scale applications



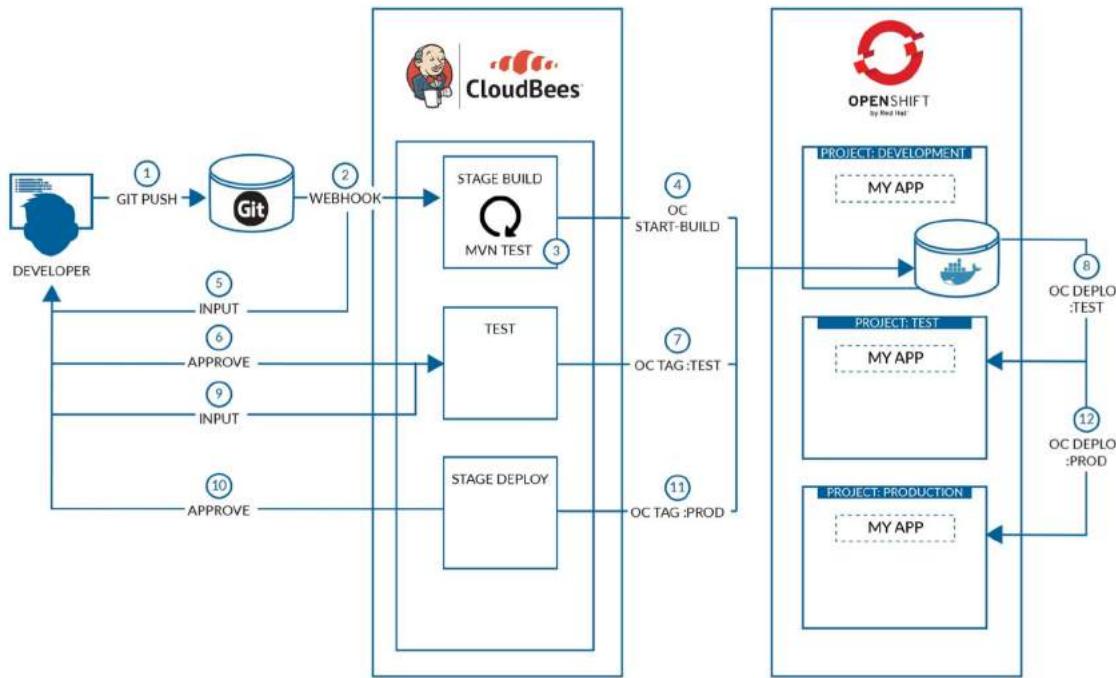
OpenShift at a glance

- PaaS
- Deploy application container from source S2I (Pod)
- Scale application pods for resiliency and load balancing
- Roll out code changes to different environments
test/staging/production/etc...
 - On-prem/Off-prem
 - Virtual or physical servers
- CI/CD pipeline integration

OpenShift Environment Architecture 101



CI/CD Pipeline

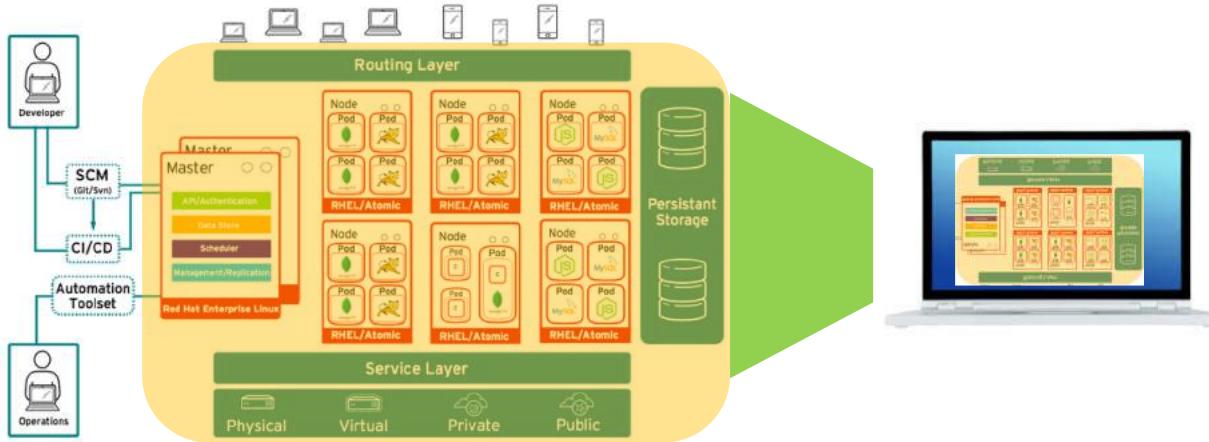


Minishift Overview

Installation and Application Deployment

Minishift is a tool that helps you run OpenShift locally by running a single-node OpenShift cluster inside a VM. You can try out OpenShift or develop with it, day-to-day, on your local host.

MiniShift Environment



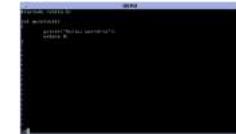
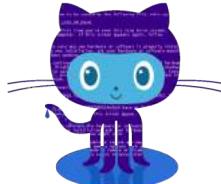
- Runs in a single VM
 - Part of Red Hat Development Suite (CDK)
 - Not Intended for production
 - Only accessible from workstation
-
- There are other ways to deploy single service OpenShift instance

Lab Notes and Links

- Google Doc
 - <http://goo.gl/m5EiAY>

MiniShift Prerequisites

- Red Hat Developer Account (Free) 
 - <https://developers.redhat.com>
- Git client and Github account (if you want to modify code examples)
 - <https://git-scm.com>
 - <https://github.com>
- IDE (PyCharm CE/Atom/Notepad++/vi/etc..)
- Basic git knowledge
 - git clone/add/commit/push



Windows Prerequisites

- virtualbox
- Tools I like that make my life better on Windows
 - Chocolatey package manager - <https://chocolatey.org>
 - Cygwin (included with RH Dev Suite)
 - ConEmu
 - For local builds
 - jdk and maven installed
 - installing maven via choco will automatically install jdk
 - python
 - git

Download and install Red Hat Development Suite

- <https://developers.redhat.com>
 - Products -> Developer Tools -> Red Hat Development Suite -> Download (<https://developers.redhat.com/products/devsuite/download>)
 - Download version 2.2 and not the latest
- Includes
 - MiniShift (MiniShift RHEL VM and startup scripts)
 - OC (OpenShift CLI)
 - VirtualBox (latest supported version)
 - Red Hat JBOSS Developer Studio (IDE)
 - cygwin
 - And more....!!!!

ALL DOWNLOADS

Version	Release Date	Description	Download
2.3.0	2018-04-22	Development Suite	Online Installer for macOS (68 MB) Installer for macOS (1 GB) Online Installer for Windows (44 MB) Installer for Windows (1 GB)

[View Older Downloads ▾](#)

ALL DOWNLOADS

Version	Release Date	Description	Download
2.2.0	2018-01-26	Development Suite	Online Installer for Windows (42 MB) Installer for Windows (1 GB) Online Installer for macOS (61 MB) Installer for macOS (1 GB)



Install location: C:\Program Files\DevelopmentSuite (Windows)
/Applications/ DevelopmentSuite (Mac)

Start Minishift

- Via command line
 - minishift start

```
➔ ~ minishift start
[-- Starting profile 'minishift'
-- Checking if requested hypervisor 'virtualbox' is supported on this platform ... OK
-- Checking if VirtualBox is installed ... OK
-- Checking the ISO URL ... OK
-- Starting local OpenShift cluster using 'virtualbox' hypervisor ...
-- Starting Minishift VM ..... OK
-- Registering machine using subscription-manager
  Red Hat Developers or Red Hat Subscription Management (RHSM) [username: leonlevy]
  Red Hat Developers or Red Hat Subscription Management (RHSM) [password: [HIDDEN]]
    Registration in progress ..... OK [14s]
-- Checking for IP address ... OK
-- Checking if external host is reachable from the Minishift VM ...
  Pinging 8.8.8.8 ... OK
-- Checking HTTP connectivity from the VM ...
  Retrieving http://minishift.io/index.html ... OK
-- Checking if persistent storage volume is mounted ... OK
-- Checking available disk space ... 61% used OK
-- OpenShift cluster will be configured with ...
  Version: v3.7.14
-- Checking `oc` support for startup flags ...
  host-config-dir ... OK
  host-data-dir ... OK
  host-pv-dir ... OK
  host-volumes-dir ... OK
  routing-suffix ... OK
Starting OpenShift using registry.access.redhat.com/openshift3/ose:v3.7.14 ...
OpenShift server started.

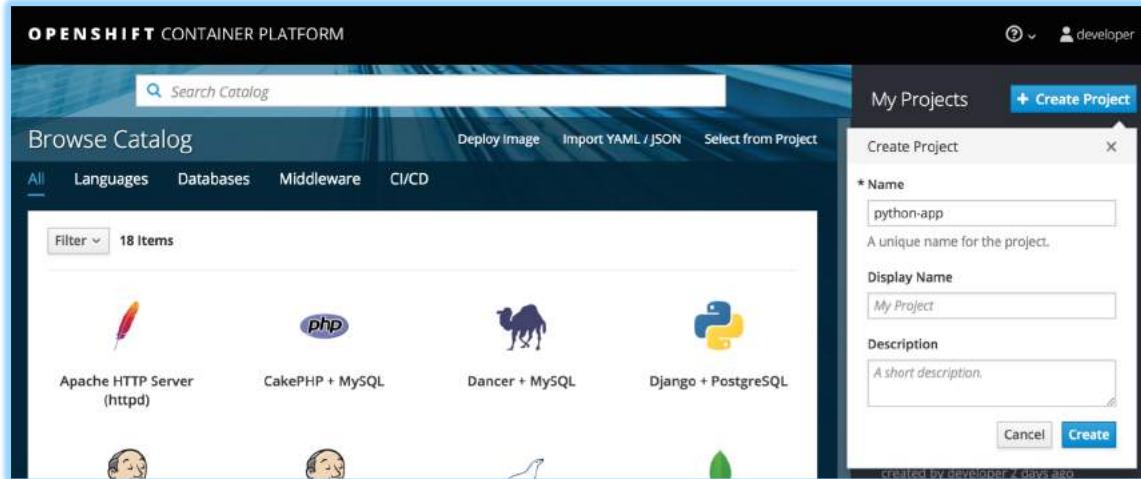
The server is accessible via web console at:
  https://192.168.99.100:8443
```

Login to MiniShift CLI and WebUI

- CLI
 - `oc login https://<ip address from startup>:8443`
- WebUI
 - Point browser to `https://<ip address from startup>:8443`
- Username/Password – developer/dev
 - `minishift ip` in case you forget IP address
 - `oc login https://$(minishift ip):8443`

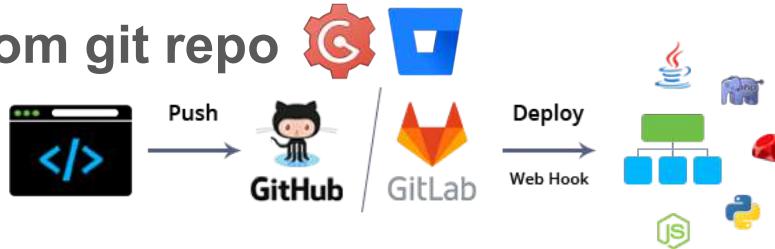
Create a new project

- CLI
 - `oc new-project python-app`
- WebUI



Create new application in project

- More than one way to do this
 - Direct from local build (maven fabric8 module in java)
 - Deploy a prebuilt container
 - **S2I from git repo**  



Sample Python/Flask Application

<https://github.com/kovarus/os-sample-python-flask>

Deploy Python application S2I

- oc new-app https://github.com/kovarus/os-sample-python-flask.git --name hello-py
- oc logs -f bc/hello-py
- oc expose service hello-py
- oc get routes

```
Pushed 6/6 layers, 100% complete
```

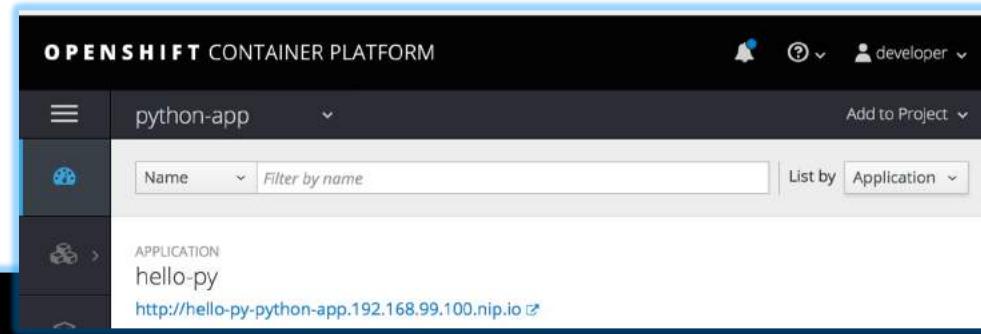
```
Push successful
```

```
→ ~ oc expose svc hello-py
```

```
route "hello-py" exposed
```

```
→ ~ oc get routes
```

NAME	HOST/PORT	PATH	SERVICES	PORT	TERMINATION	WILDCARD
hello-py	hello-py-python-app.192.168.99.100.nip.io		hello-py	8080-tcp		None



Python App

The screenshot shows a web browser window with the following details:

- Tab Bar:** kovarus/os-sample-python | OpenShift Web Console | Flask on OpenShift - Exam
- Address Bar:** hello-py-python-app.192.168.99.100.nip.io
- Content Area:**
 - Hello, User!**
 - Running in host/container: hello-py-1-x9jgh
 - It is now: 06-19-2018 15:49.
- Code Preview:** 12 lines (11 sloc) | 259 Bytes

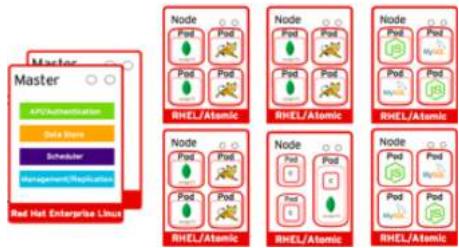
Code Preview Content:

```
15
16 @application.route('/')
17 @application.route('/index')
18 def index():
19     now = datetime.datetime.now()
20     user = {'username': 'User'}
21     hostname = socket.gethostname()
22     return render_template('index.html', title='Flask on OpenShift',
23                           user=user,
24                           hostname=hostname,
25                           time=now.strftime('%m-%d-%Y %H:%M'))
```

Code Preview Content (Continued):

```
1  <html>
2      <head>
3          <title>{{ title }} - Example</title>
4      </head>
5      <body>
6          <h1>Hello, {{ user.username }}!</h1>
7          <p>Running in host/container: {{ hostname }}</p>
8          <br>
9          <p>It is now: {{ time }}.</p>
10     </body>
11 </html>
```

Scaling application



The screenshot shows the OpenShift Container Platform interface. The top navigation bar says 'OPENShift CONTAINER PLATFORM' and has a user icon for 'developer'. There are buttons for 'Add to Project' and 'List by Application'. The main area shows an application named 'hello-py' with its URL: <http://hello-py-python-app.192.168.99.100.nip.io>. Below this is a deployment named 'hello-py, #1' with a status of '1 pod'. A large blue callout box highlights the 'hello-py' section, which shows a summary of the deployment: 'hello-py' and its URL. It also shows the container details: 'CONTAINER: HELLO-PY' with 'Image: python-app/hello-py 7a422d1 214.3 MB', 'Build: hello-py, #1', 'Source: Update README.md 3dbb571', and 'Ports: 8080/TCP'. A circular icon indicates '3 pods'.

```
oc scale --replicas=3 dc hello-py
```

Deploy Build (i.e. code changes)

- `oc start-build hello-py`
- `oc logs -f bc/hello-py` (**track build progress**)

APPLICATION
hello-py

http://hello-py-python-app.192.168.99.100.nip.io ↗

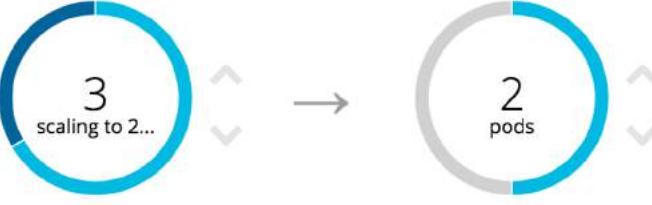
DEPLOYMENT
hello-py, #4

Rolling deployment is running... [View Events](#) | [Cancel](#)

3 scaling to 2...

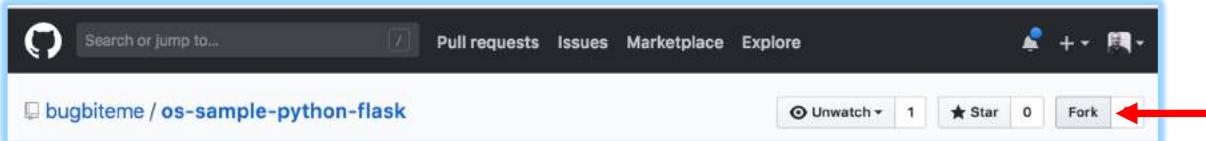
2 pods

Networking



Code change example (using git) Slide 1 of 3

- Go to <https://github.com/kovarus/os-sample-python-flask>
- Fork project to your own account



- Deploy new project and app from forked repo

```
oc new-project python-app2  
oc new-app https://github.com/<your repo>/os-sample-python-flask.git -name hello-py  
oc expose service hello-py  
oc get routes
```
- Make sure app deployed as expected (view via web browser)
- Scale to 3 instances
 - `oc scale --replicas=3 dc hello-py`

Code change example (using git) Slide 2 of 3

- Clone forked repo to local system
 - `git clone https://github.com/<your repo>/os-sample-python-flask.git`
- Change some code!
 - `cd os-sample-python-flask`
- Edit `wsgi.py` (`vi`, `notepad++`, `PyCharm`, `Atom`, etc...)

```
14
15
16     @application.route('/')
17     @application.route('/index')
18     def index():
19         now = datetime.datetime.now()
20         user = {'username': 'State of CA'}
21         hostname = socket.gethostname()
22         return render_template('index.html', title='Flask on OpenShift',
23                               user=user,
24                               hostname=hostname,
25                               time=now.strftime('%m-%d-%Y %H:%M'))
26
```

Code change example (using git) Slide 3 of 3

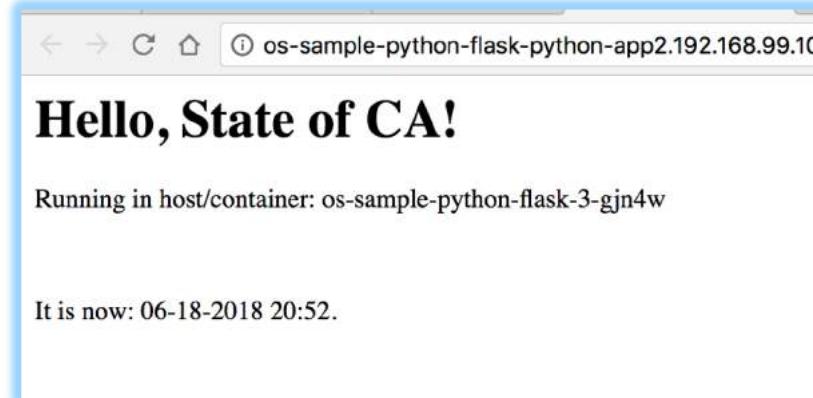
- Save code and check changes into github repo

```
git add *
git commit -m "doing some code changes"
git push
```

- Build and deploy code changes

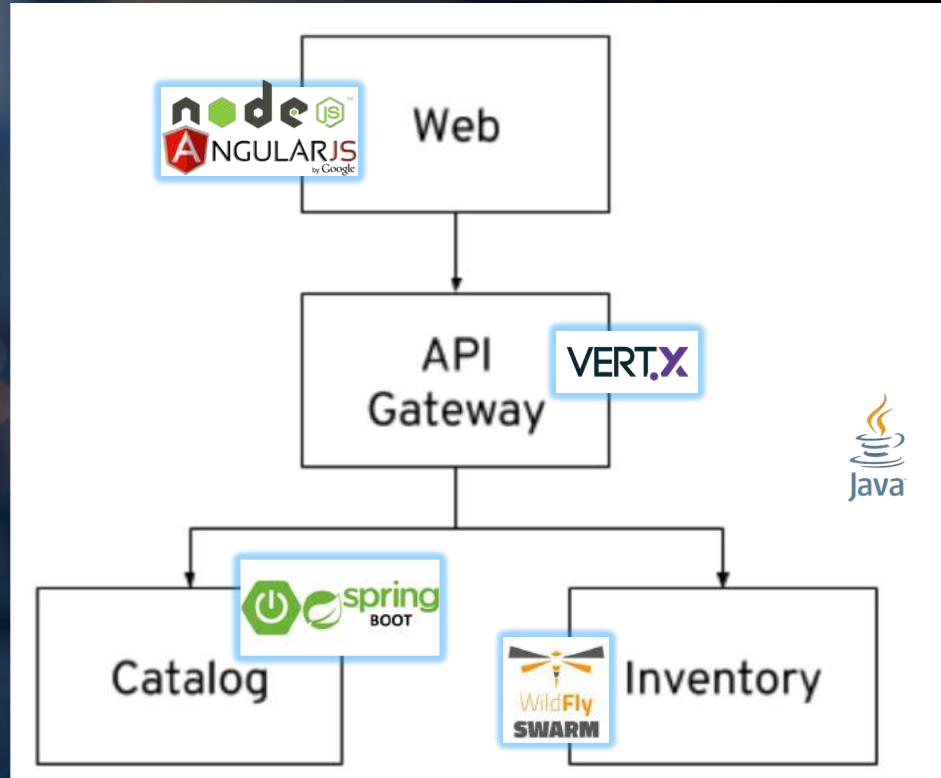
```
oc start-build hello-py; oc logs -f bc/hello-py
```

- Refresh web-browser to view changes



Java Microservice Application

Demo



Components – Coolstore App (Kovarus Edition)

- **Inventory** - REST API for obtaining inventory from DB using Wildfly Swarm Java framework
 - <http://wildfly-swarm.io>
- **Catalog** - REST API for obtaining inventory from DB using Spring Boot Java framework
 - <https://spring.io/projects/spring-boot>
- **API Gateway** – Gateway for Inventory and Catalog using vert.x Java framework
 - <https://vertx.io>
- **Web** – Web frontend to end-user using NodeJS and AngularJS
 - <https://nodejs.org>
 - <https://angular.io>

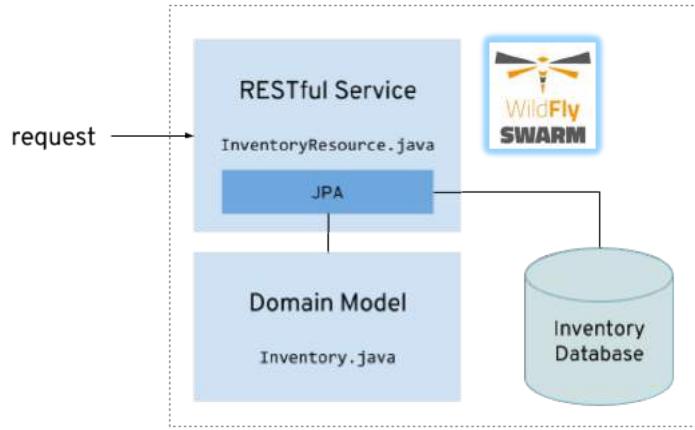
Easy to follow instructions to deploy it yourself!

Component	Github repo
Inventory	https://github.com/kovarus/inventory-wildfly-swarm
Catalog	https://github.com/kovarus/catalog-spring-boot
API Gateway	https://github.com/kovarus/gateway-vertx
Web	https://github.com/kovarus/web-nodejs



Inventory

- <https://github.com/kovarus/inventory-wildfly-swarm>

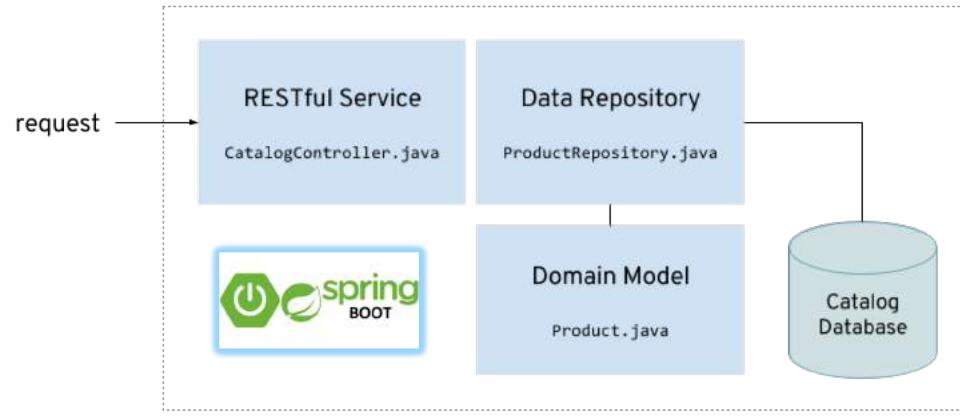


<http://inventory-coolstore.192.168.99.100.nip.io/api/inventory/329299>

```
{  
    "itemId": "329299",  
    "quantity": 35  
}
```

Catalog

- <https://github.com/kovarus/catalog-spring-boot>

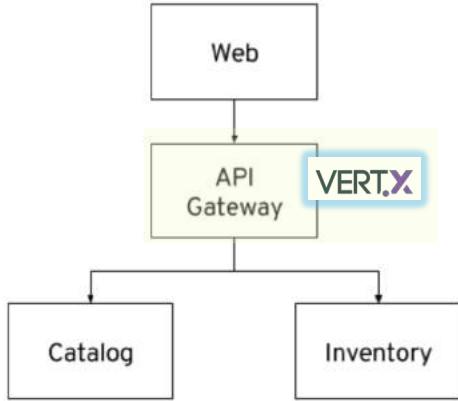


<http://catalog-coolstore.192.168.99.100.nip.io/api/catalog>

```
[  
 {  
   "itemId": "329299",  
   "name": "Red Fedora",  
   "description": "Official Red Hat Fedora",  
   "price": 34.99  
 },  
 {  
   "itemId": "329199",  
   "name": "Forge Laptop Sticker", "...
```

API Gateway

- <https://github.com/kovarus/gateway-vertx>



<http://gateway-coolstore.192.168.99.100.nip.io/api/products>

```
[{"id": 1, "name": "Red Fedora", "description": "Official Red Hat Fedora", "price": 34.99, "availability": {"quantity": 35}}, {"id": 2, "name": "Solid Performance Polo", "description": "Moisture-wicking, antimicrobial 100% polyester d", "price": 17.8, "availability": {"quantity": 45}}, {"id": 3, "name": "750ml Camelbak Water Bottle", "description": "Redesigned cap and bite valve provide faster flo closure; for cold beverages only. Holds 25 oz. Hand wash only. I", "price": 6, "availability": {"quantity": 43}}]
```

Web frontend

- <https://github.com/kovarus/web-nodejs>
- <http://web-coolstore.192.168.99.100.nip.io>

The screenshot shows a web-based shopping cart interface for 'Kovarus Cool Store'. The top navigation bar includes the store logo, a shopping cart icon with '\$0.00 (0 item(s))', and a 'Sign In' link. The main content area displays three products:

- Ogio Caliber Polo**: Description: Moisture-wicking 100% polyester. Rib-knit collar and cuffs; Ogio jacquard tape insitem_id neck; bar-tacked three-button placket with...
Image: A man wearing a green Ogio Caliber Polo shirt.
Price: \$28.75
Buttons: 'Add To Cart' (with quantity dropdown), '87 left!', and a location pin icon.
- Red Fedora**: Description: Official Red Hat Fedora
Image: Two images: one of a red fedora hat and another showing a person wearing it.
Price: \$34.99
Buttons: 'Add To Cart' (with quantity dropdown), '35 left!', and a location pin icon.
- 750ml Camelbak Water Bottle**: Description: Redesigned cap and bite valve provide faster flow and enhanced durability; BPA-free, acrylic bottle. Push-on item_id with thumb-slitem_id closure; for cold beverages only. Holds 25 oz. Hand wash only. Imprint. Grey or Blue.
Image: A black Camelbak water bottle with a blue 'KOVARUS' logo and a blue Camelbak water bottle with a white 'A' logo.
Buttons: None shown for this item.

For more information...

- OpenShift Container Platform 3.7 Documentation
 - <https://docs.openshift.com/container-platform/3.7>
- OpenShift Cloud-native Roadshow
 - <http://guides-cdk-roadshow.b9ad.pro-us-east-1.openshiftapps.com>
- Kovarus Github repos (includes all the code seen here)
 - <https://github.com/kovarus>
- Full-blown coolstore app source code and instructions
 - <https://github.com/jamesfalkner/coolstore-microservice-brms>

Advanced Topics

- Monitoring Application Health
- Service Resilience and Fault Tolerance
- Managing Application Configuration
- Automating Deployments Using Pipelines (future)
- Debugging Applications (future)
- Twelve Factor Applications

Monitoring Application Health – coolstore Example

- **Liveness Probe**

- Checks if the container in which it is configured is still running. If the liveness probe fails, the kubelet kills the container, which will be subjected to its restart policy.

- **Readiness Probe**

- A readiness probe determines if a container is ready to service requests. If the readiness probe fails a container, the endpoints controller ensures the container has its IP address removed from the endpoints of all services.

https://docs.openshift.com/container-platform/3.7/dev_guide/application_health.html#container-health-checks-using-probes

The image shows four browser windows illustrating application health monitoring:

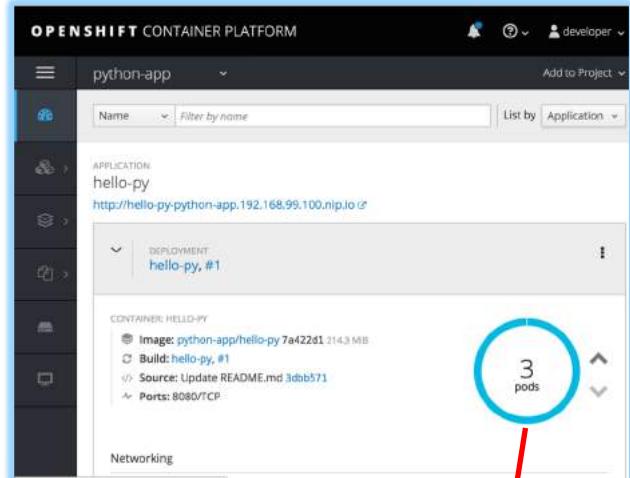
- gateway-coolstore.192.168.99.100.nip.io/health**: Returns a simple JSON object with a single key "status" set to "UP".
- catalog-coolstore.192.168.99.100.nip.io/health**: Returns a JSON object containing "status": "UP", "diskSpace": { "status": "UP", "total": 10725883904, "free": 10164224000, "threshold": 10485760 }, "db": { "status": "UP", "database": "H2", "hello": 1 }, and "refreshScope": { "status": "UP" }.
- inventory-coolstore.192.168.99.100.nip.io/node**: Returns a JSON object with a single key "name" pointing to "inventory-2-mkq67".
- inventory-coolstore.192.168.99.100.nip.io/node**: Returns a JSON object with fields: "name": "inventory-2-mkq67", "server-state": "running", "suspend-state": "RUNNING", "running-mode": "NORMAL", "uuid": "76dbeab8-669a-475d-85c1-61fd6da5e561", and "swarm-version": "2017.8.1".

Service Resilience and Fault Tolerance

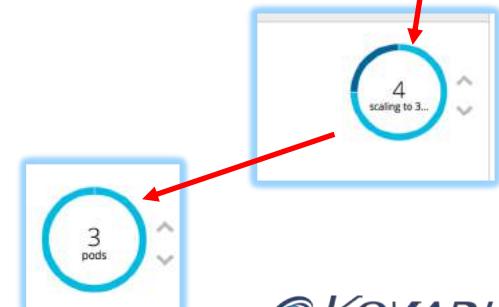
- Ensure you are back in the python-app2 project
 - `oc project python-app2`
- Ensure there are 3 containers/pods running
 - `oc scale --replicas=3 dc hello-py`
- Run command to view pods
 - `oc get pods`

```
➜ ~ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
hello-py-12-jxs8f  1/1     Running   0          1h
hello-py-12-l8q6z  1/1     Running   0          42m
hello-py-12-ppk6z  1/1     Running   0          1h
```

- Delete one of the pods running
 - `oc delete pod hello-py-<build id>`
- `oc delete pod hello-py-12-jxs8f`
`pod "hello-py-12-jxs8f" deleted`
- Observe self-healing (via Web GUI)



The screenshot shows the OpenShift web interface for the 'python-app' project. The 'hello-py' deployment is selected, displaying details such as the container image (python-app/hello-py 7a422d1), build ID (#1), source (README.md), and ports (8080/TCP). A summary indicates 3 pods.



Managing Application Configuration

Set environment variables at time of application creation or post-deployment

```
$ oc new-app postgresql-persistent \
--param=DATABASE_SERVICE_NAME=inventory-postgresql \
--param=POSTGRESQL_DATABASE=inventory \
--param=POSTGRESQL_USER=inventory \
--param=POSTGRESQL_PASSWORD=inventory \
--labels=app=inventory
```

Using config map *.yml file

```
$ cat <<EOF > ./project-stages.yml
project:
  stage: prod
swarm:
  datasources:
    data-sources:
      InventoryDS:
        driver-name: postgresql
        connection-url: jdbc:postgresql://inventory-postgresql:5432/inventory
        user-name: inventory
        password: inventory
EOF
```

```
$ oc create configmap inventory --from-file=./project-stages.yml
```

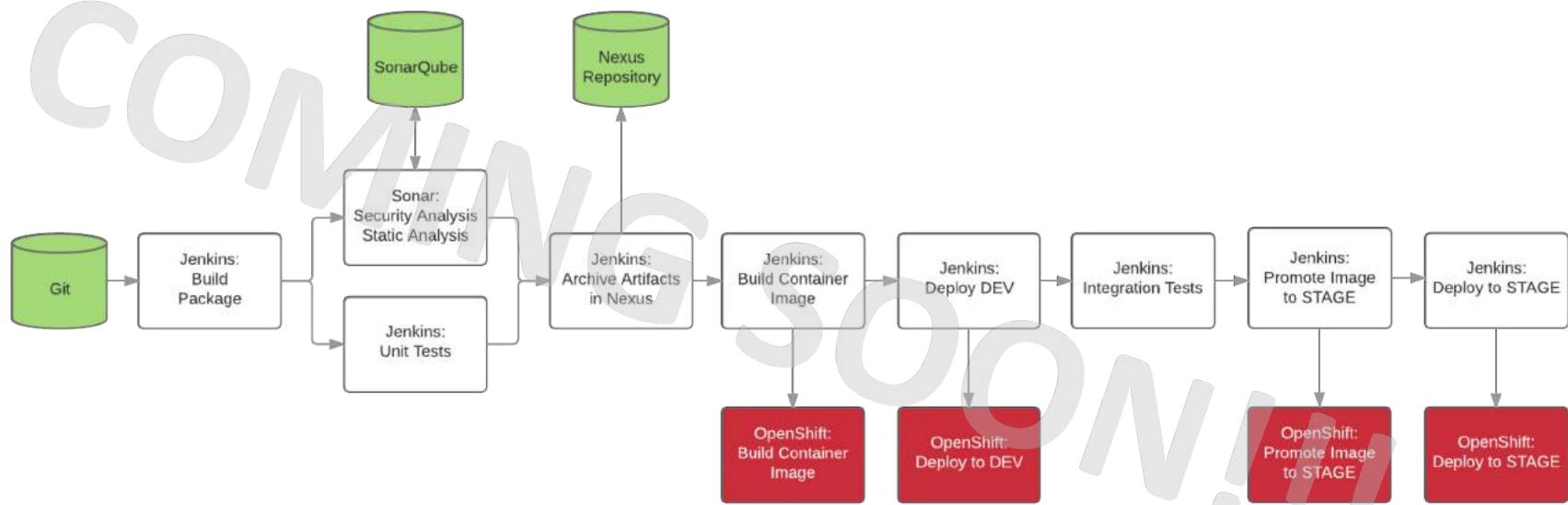
GUI Interface

The screenshot shows the OpenShift Container Platform interface. In the top navigation bar, the project 'python-app2' is selected. Below it, the application 'hello-py' is shown, which was created 2 hours ago. The 'Environment' tab is currently active. Under the 'Container hello-py' section, there is a table with one row. The 'Name' column contains 'STATE' and the 'Value' column contains '"California"'. There are buttons for 'Add Value' and 'Add Value from Config Map or Secret'.

Name	Value
STATE	"California"

Add Value | Add Value from Config Map or Secret

Automating Deployments Using Pipelines



<https://github.com/OpenShiftDemos/openshift-cd-demo>

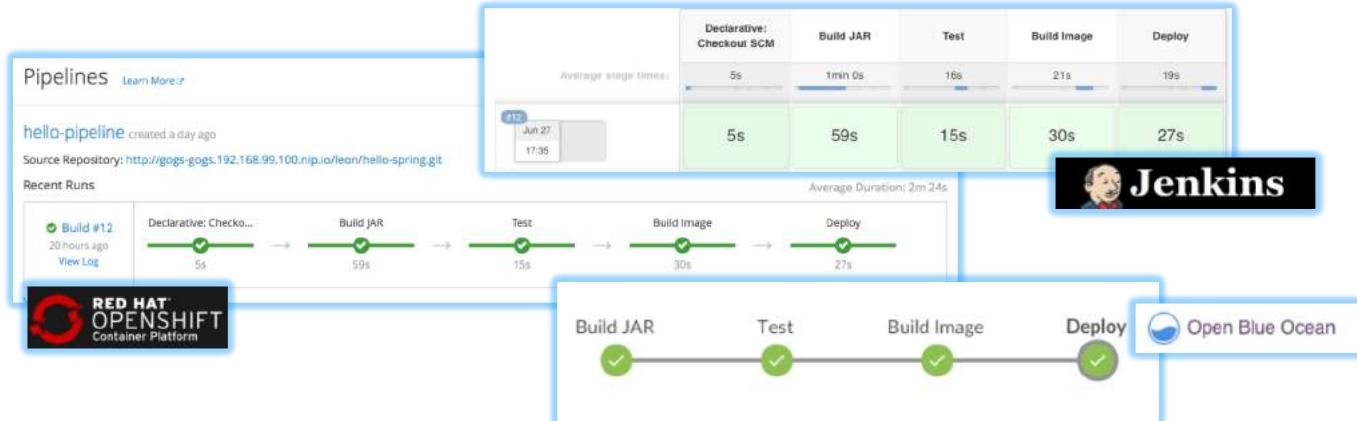
In the meantime:

<https://github.com/kovarus/hello-spring>

Simple pipeline example

<https://github.com/kovarus/hello-spring>

- Set up git repo inside OpenShift
- Clone public repo to local repo
- Deploy application from local git repo
- Set up pipeline (Jenkins)
- Webhook configuration for auto deploy after code push to git
- Unit testing



Twelve Factor Applications - <https://12factor.net>

I. Codebase

One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Config

Store config in the environment

IV. Backing services

Treat backing services as attached resources

V. Build, release, run

Strictly separate build and run stages

VI. Processes

Execute the app as one or more stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep development, staging, and production as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin/management tasks as one-off processes



THANK YOU!