OpenShift 4

For Developers

Wanja Pernath
wpernath@redhat.com

EMEA Partner Enablement Manager, OpenShift & MW

Red Hat

Self introduction

Name: Wanja Pernath

Email: wpernath@redhat.com

Base: Germany (very close to the Alps)

Role: EMEA Technical Partner Development Manager - OpenShift and MW

Experience: Years of Consulting, Training, PreSales at Red Hat and before

OpenShift Container Platform

Advanced Cluster Management

Multi-cluster Management

Discovery ⁝ Policy ⁝ Compliance ⁝ Configuration ⁝ Workloads

OpenShift Container Platform

Manage Workloads	Build Cloud-Native Apps	Developer Productivity
Platform Services	**Application Services**	**Developer Services**
Service Mesh ⁝ Serverless Builds ⁝ CI/CD Pipelines Full Stack Logging Chargeback	Databases ⁝ Languages Runtimes ⁝ Integration Business Automation 100+ ISV Services	Developer CLI ⁝ VS Code extensions ⁝ IDE Plugins Code Ready Workspaces CodeReady Containers

OpenShift Kubernetes Engine

Cluster Services

Automated Ops ⁝ Over-The-Air Updates ⁝ Monitoring ⁝ Registry ⁝ Networking ⁝ Router ⁝ KubeVirt ⁝ OLM ⁝ Helm

Kubernetes

Red Hat Enterprise Linux & RHEL CoreOS


Physical


Virtual


Private cloud


Public cloud


Managed cloud
(Azure, AWS, IBM, Red Hat)

Red Hat

Web Console

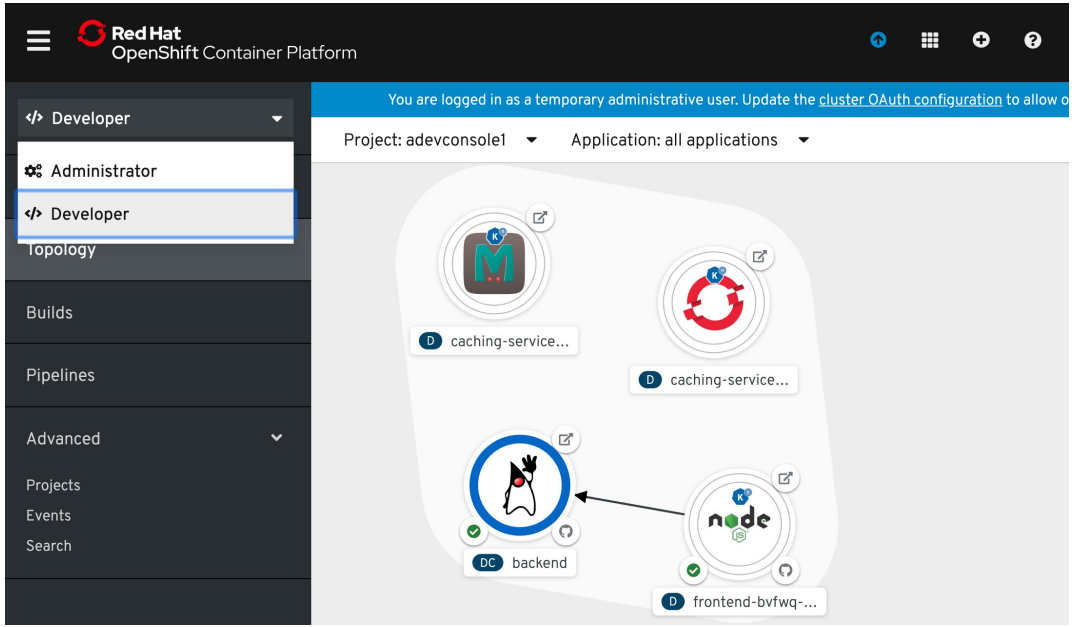
Photo by Unsplash

Red Hat

OpenShift Developer Console

An developer-specific perspective in the OpenShift UI that will sit beside the admin console and focus on developer use cases.

All OpenShift developer tool UIs will be surfaced here.

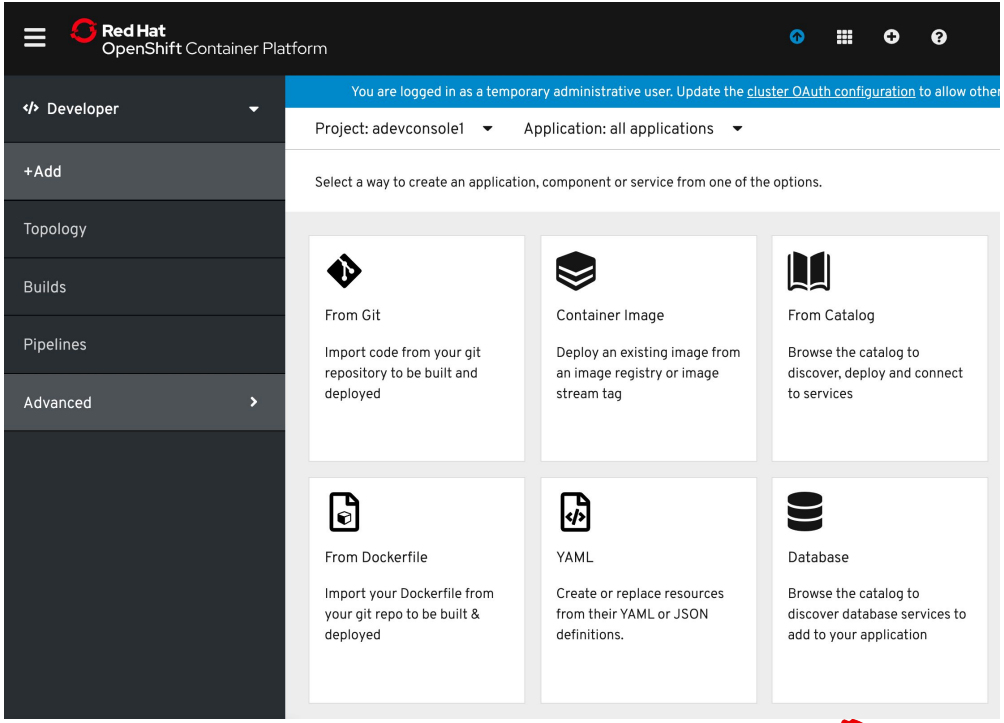


Red Hat

Developer Console: Create Applications

Key Features

- Import source from Git
- View existing container image
- Edit YAML definition
- Build from Dockerfile
- Explore services catalog
- Deploy database from catalog

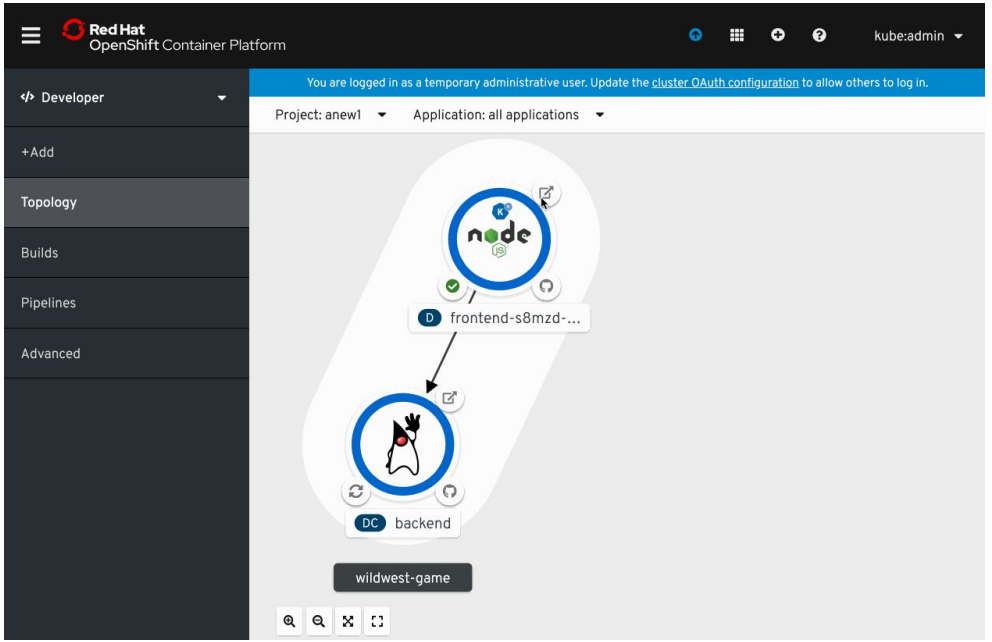


Red Hat

Developer Console: Application Topology

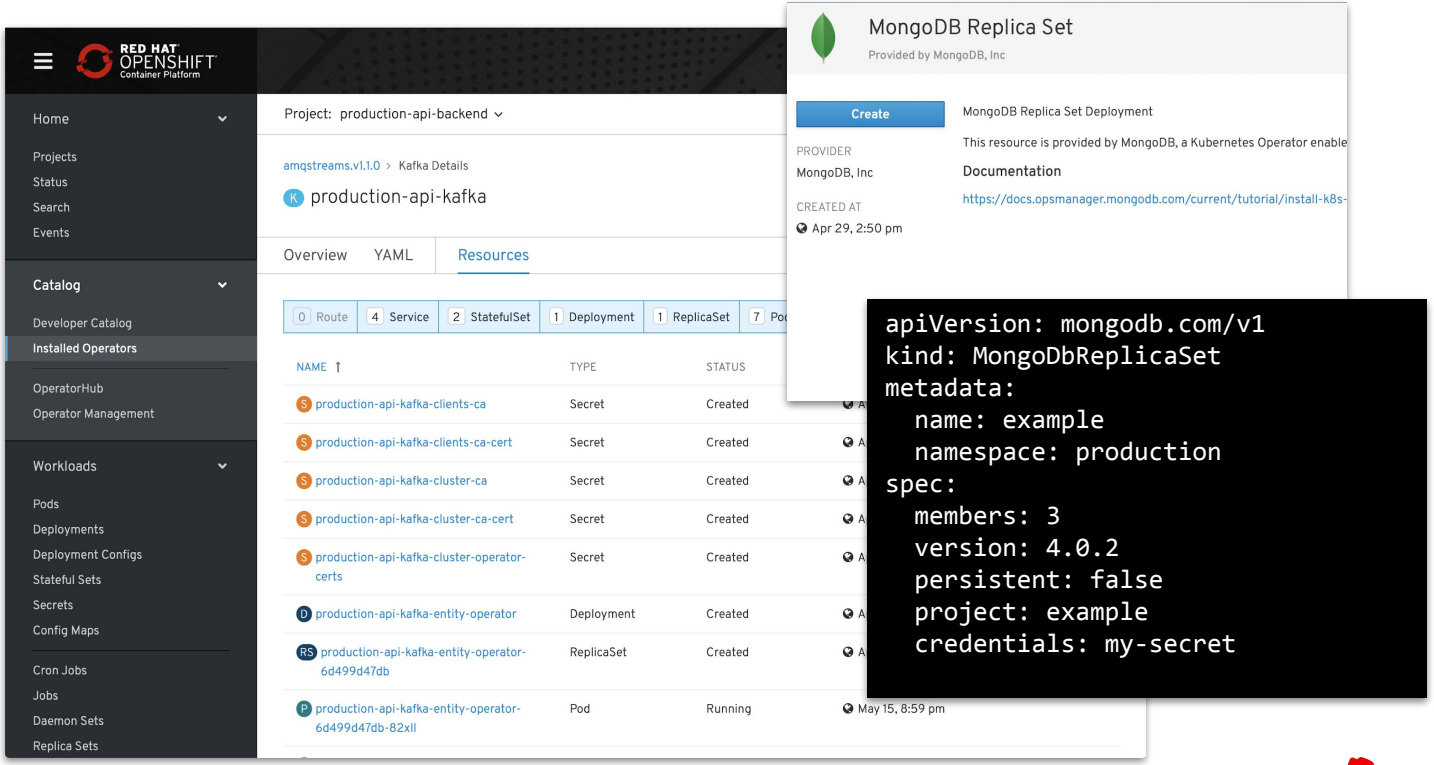
Key Features

- View structure and status of app components
- Drill into specific workloads
- Quickly navigate to pod logs
- Manually scale
- Pod donut!
- Access route/URL
- Linked build and source



Red Hat

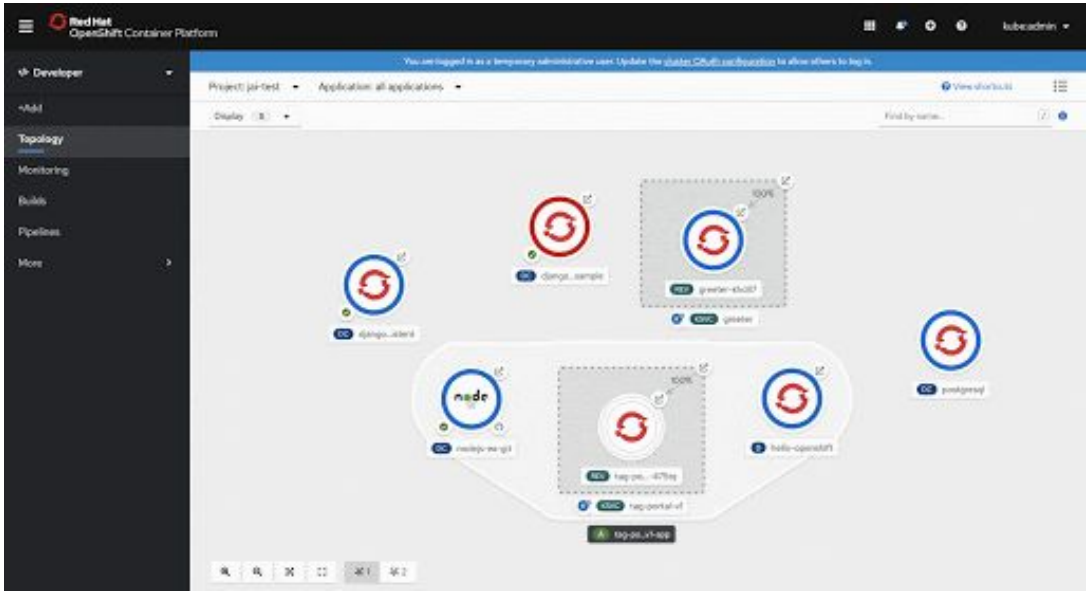
Self-service for developers



Red Hat

Understanding Complex Apps find

- Allow developers to easily find and focus on interested components
- Matched items are highlighted and non-matches are dimmed
- Indicates matches off of visible area

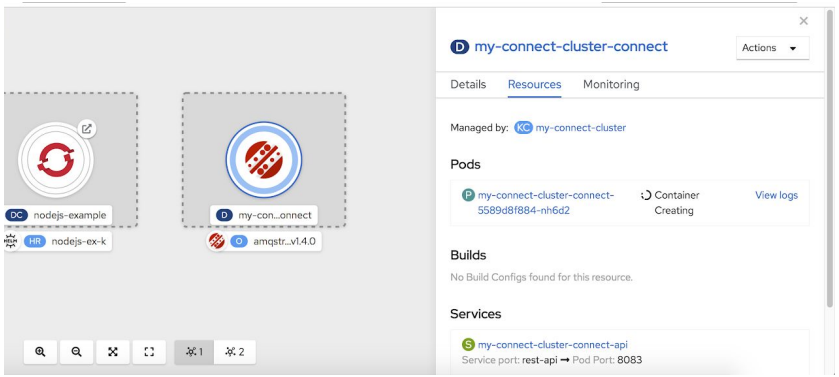


Red Hat

Understanding Complex Apps grouping, collapse

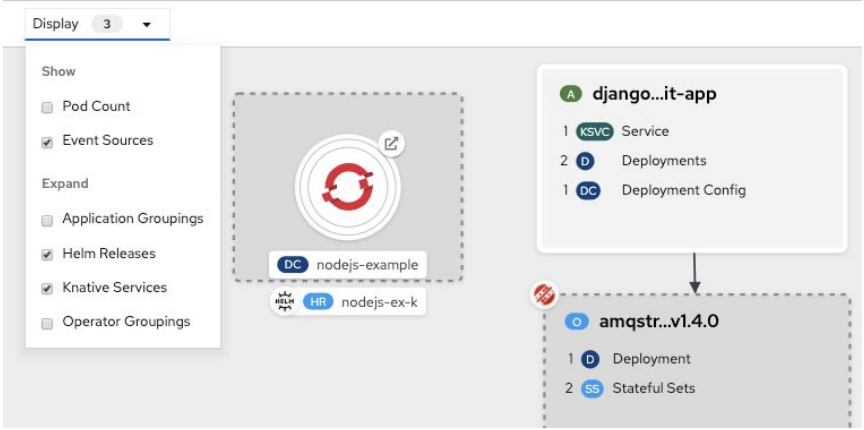
Operator-managed and Helm-based services

- Groups all resources to easily understand what is being managed by which CR/operator
- Add simple info in resources side panel, for custom resource or general resource parent



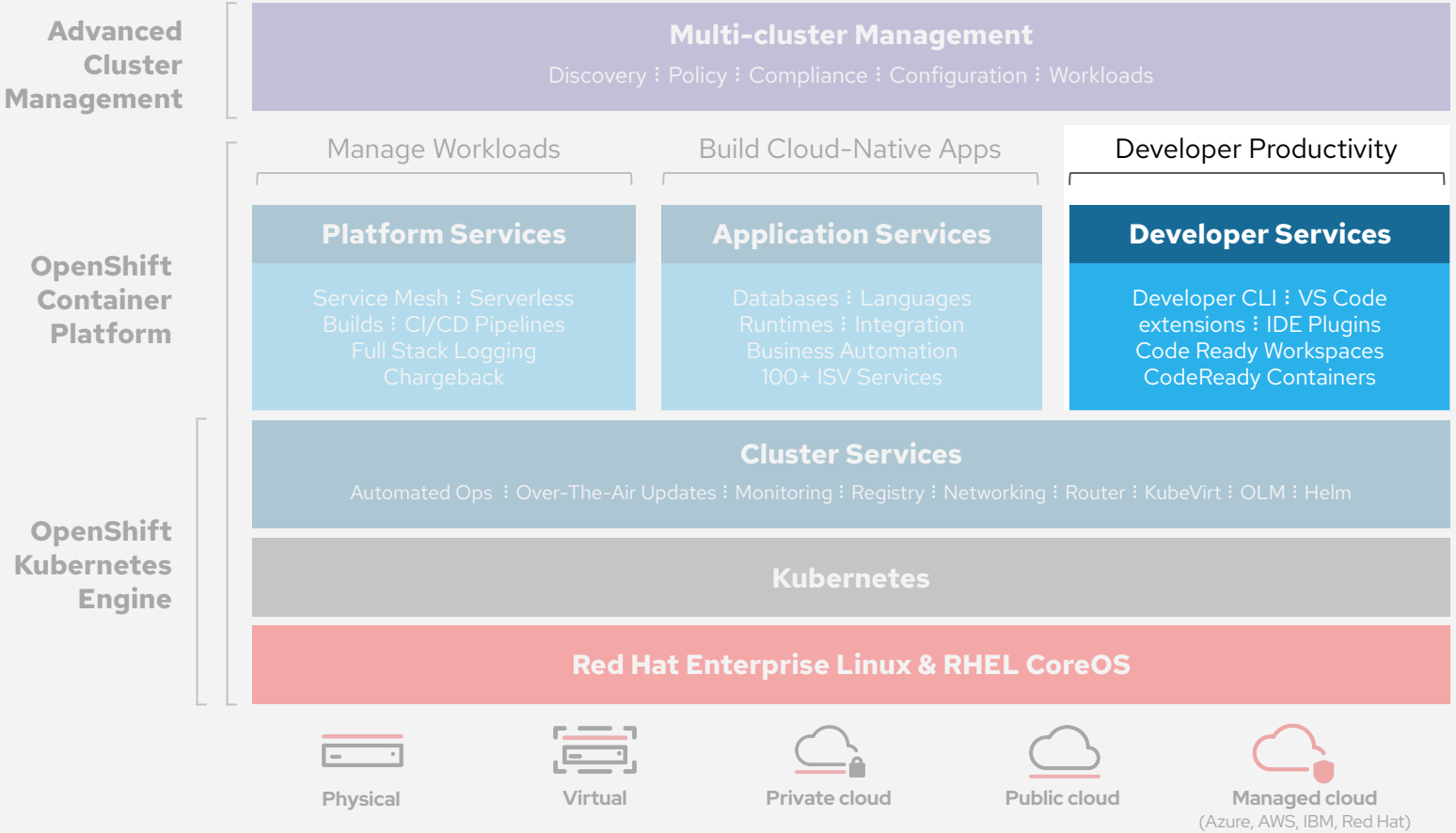
Collapse / Expand

- Manage large multi-application projects by expanding/collapsing certain grouping types
- Reduce Application, Helm, Knative and Operator to simple summary tiles



Red Hat

OpenShift Container Platform



Red Hat

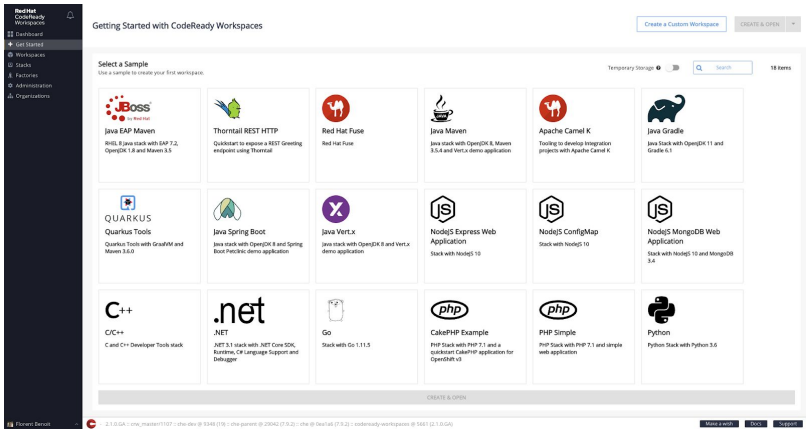
Desktop Tools, IDEs & Plugins

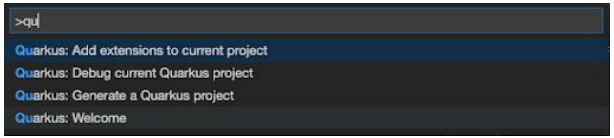
Photo by Unsplash

Red Hat

CodeReady Workspaces 2.x

- New onboarding experience – updated catalog and simple workspace creation options
- Quarkus support – includes stack and plugin
- Updated languages – .NET 3.1, Camel K, Java 11 & Gradle
- Air Gap improvements
- Many editor enhancements





Red Hat

odo - OpenShift's Dev-Focused CLI

Developer-focused CLI for rapid development iterations on OpenShift

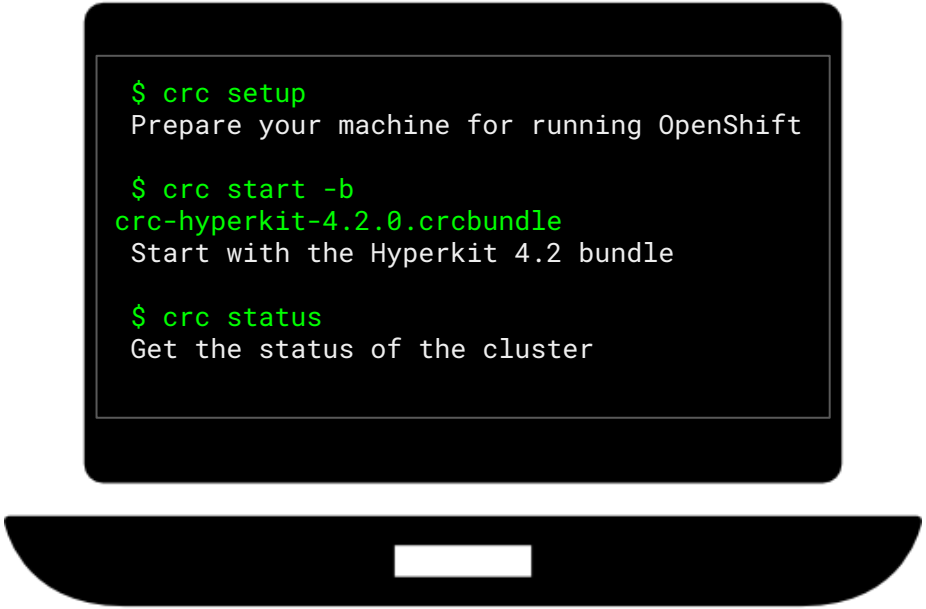
Simplifies building of microservices applications on OpenShift.

```
$ odo create wildfly backend
Component 'backend' was created.

$ odo push
Pushing changes to component: backend

$ odo create php frontend
Component 'frontend' was created.
To push source code to the component run 'odo push'

$ odo push
Pushing changes to component: frontend

$ odo url create
frontend - http://frontend-myapp.192.168.99.100.nip.io

$ odo watch
Waiting for something to change in /dev/frontend
```

CodeReady Containers

Environment to simplify local direct-to-OpenShift application development.

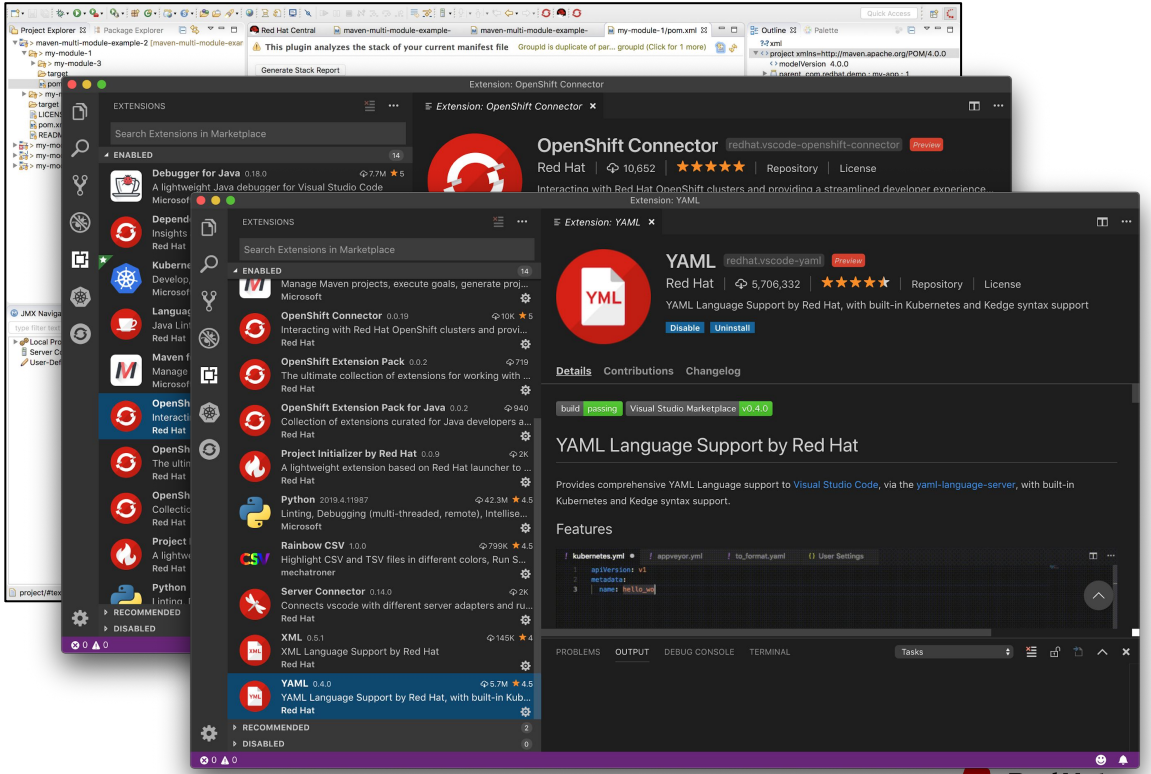
Available for:
- Linux (KVM)
- Windows (Virtualbox)
- MacOS (Virtualbox)



IDE Plugins for OpenShift

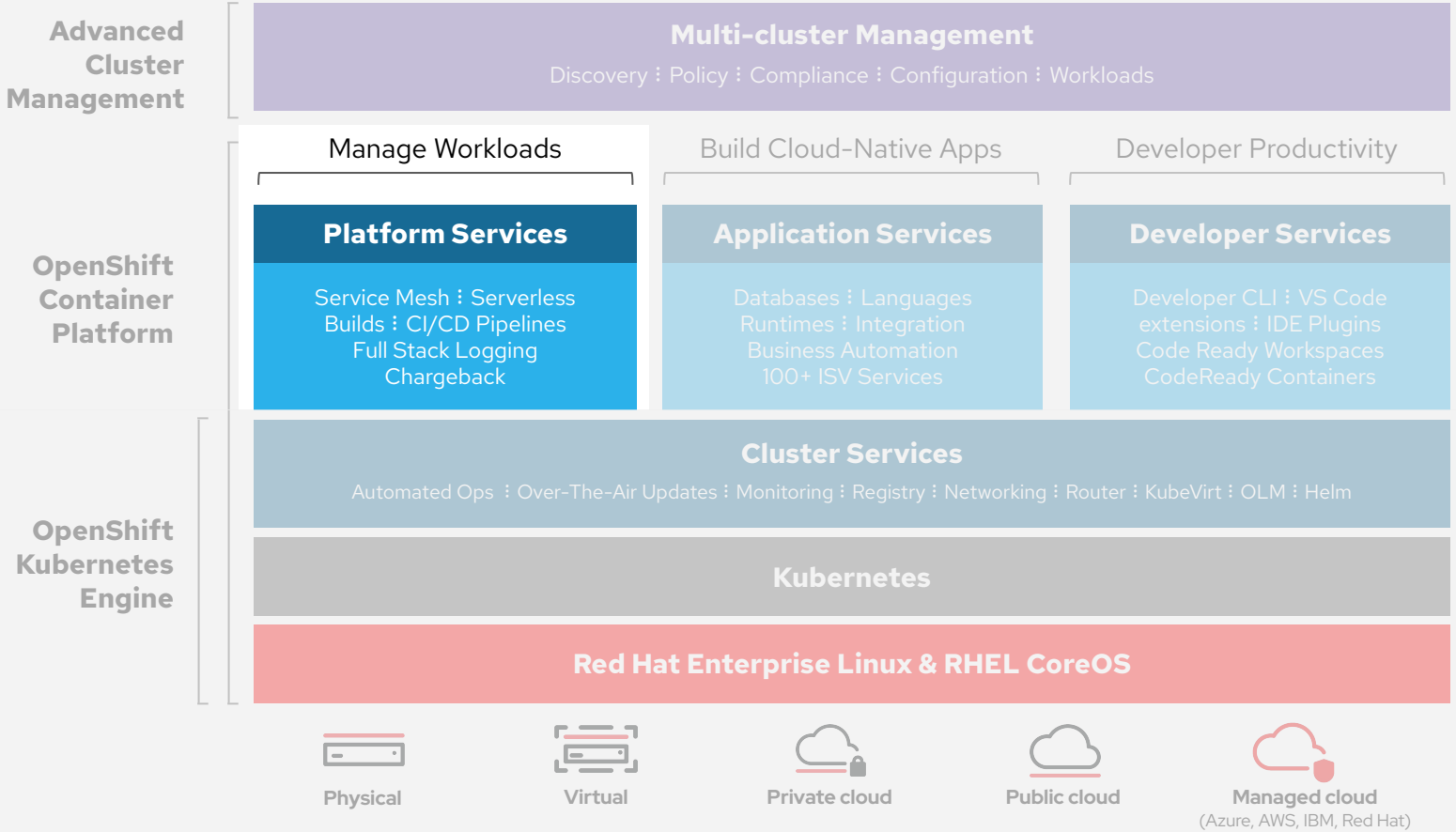
We have a suite of plugins to simplify working with OpenShift and YAML, as well as our Middleware products such as Java, Camel, Wildfly, and much more. These extensions are available for:

- Azure DevOps
- VS Code
- JetBrains IDEs (e.g. IntelliJ)
- CodeReady Workspaces
- CodeReady Studio
- Eclipse IDE/JBoss Tools



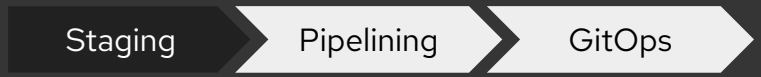
Red Hat

OpenShift Container Platform



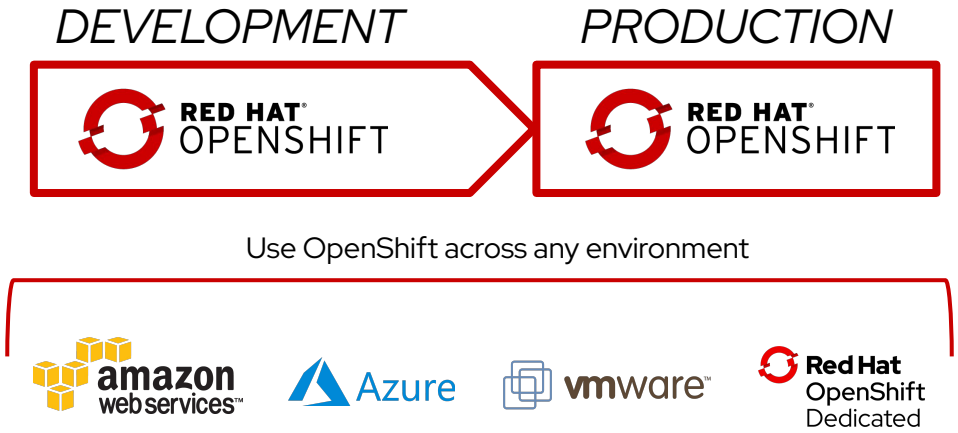
Red Hat

From Code to Image



Red Hat

Red Hat OpenShift
Adds consistency and portability to your cloud journey



*** Develop on OpenShift for existing apps and new cloud-native apps**

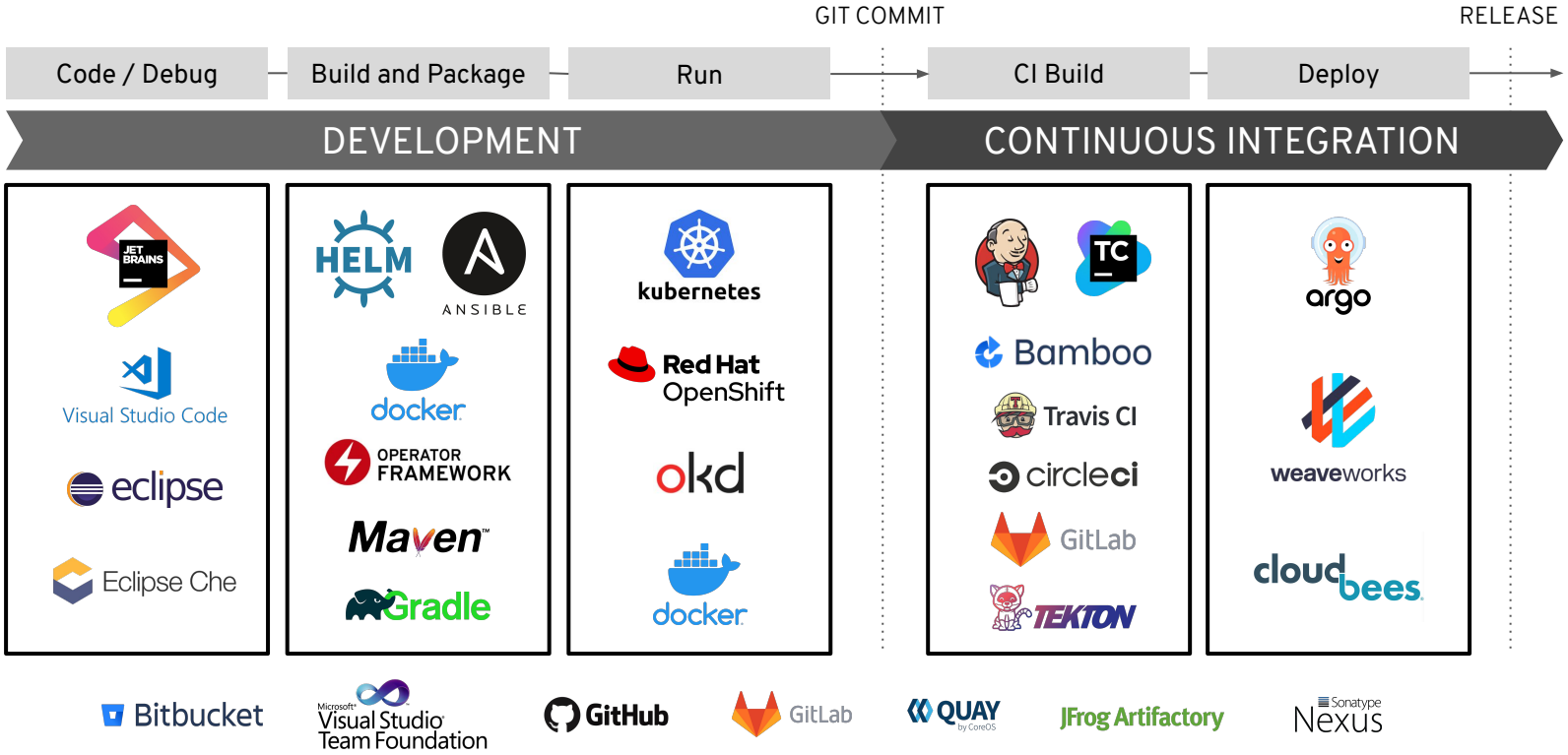
*** Deploy on OpenShift in AWS, Azure, current environments or anywhere else**

One consistent development environment enables developers to move from project to project quickly. They don't need to learn new cloud-specific UIs and tools.

If teams need to leverage other cloud providers at any point it's easy to move specific apps because OpenShift provides an abstraction layer.

Red Hat

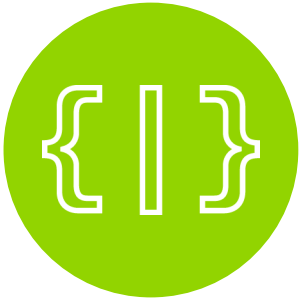
OpenShift integrates into your organization's preferred toolchain



Red Hat

BUILD & DEPLOY

BUILD AND DEPLOY CONTAINER IMAGES

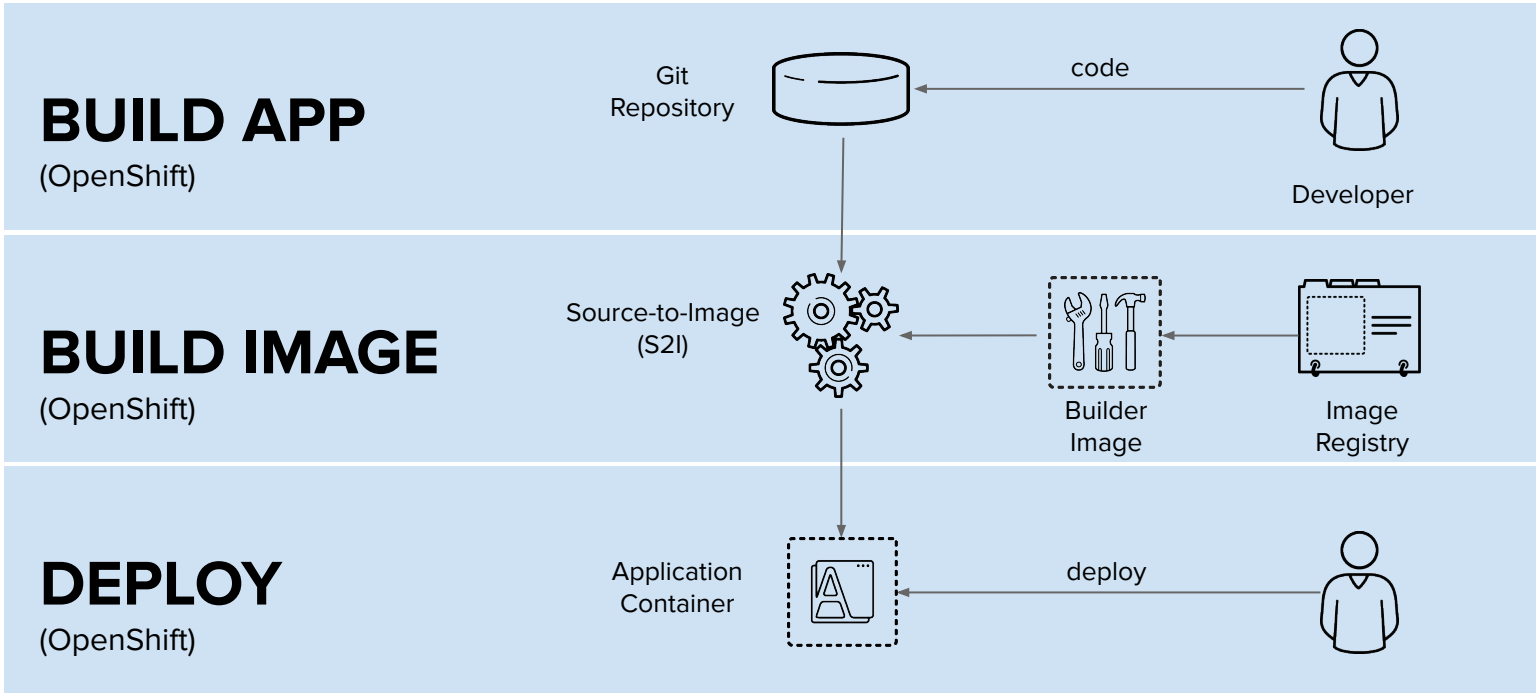


DEPLOY YOUR SOURCE CODE

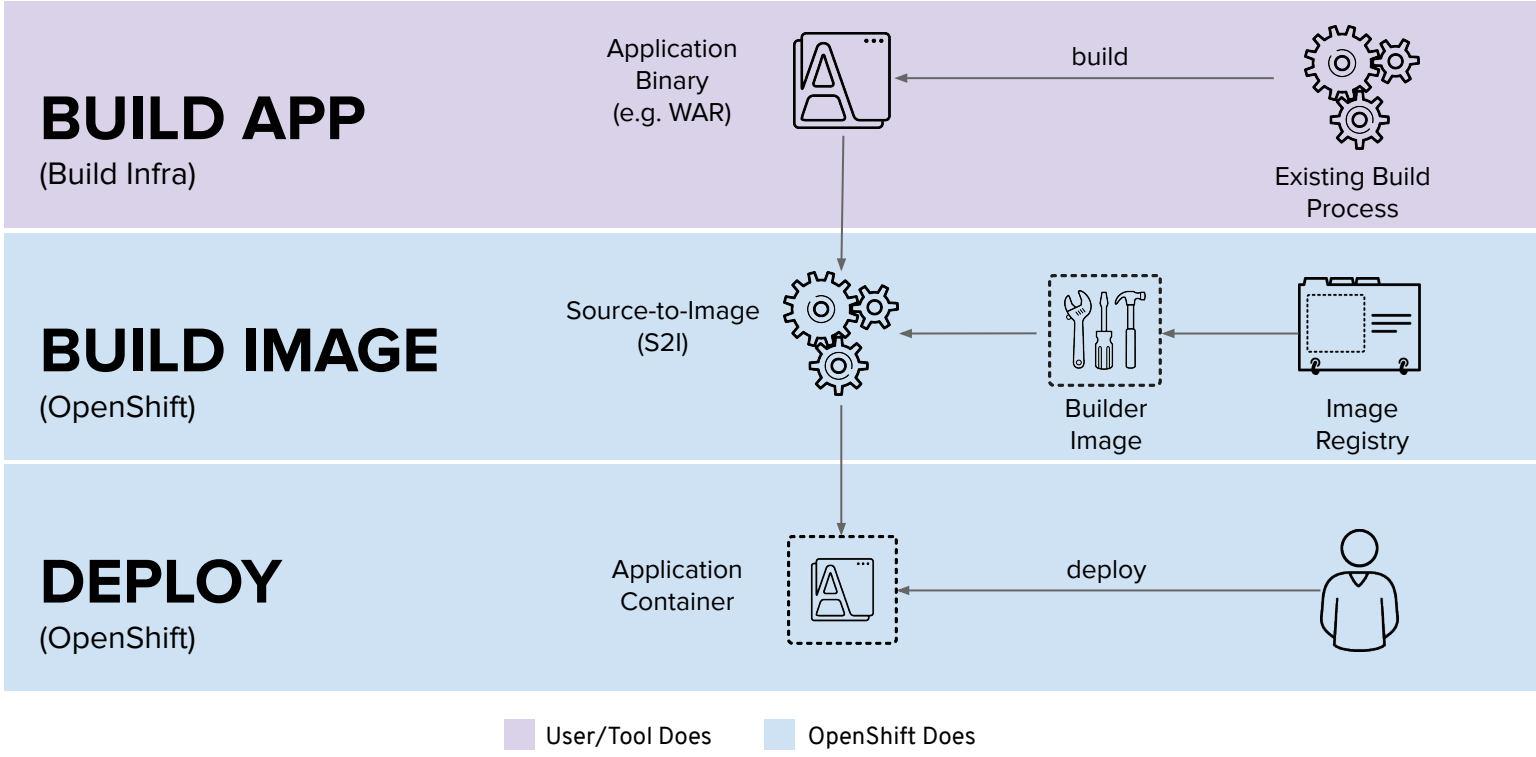
DEPLOY YOUR APP BINARY

DEPLOY YOUR CONTAINER IMAGE

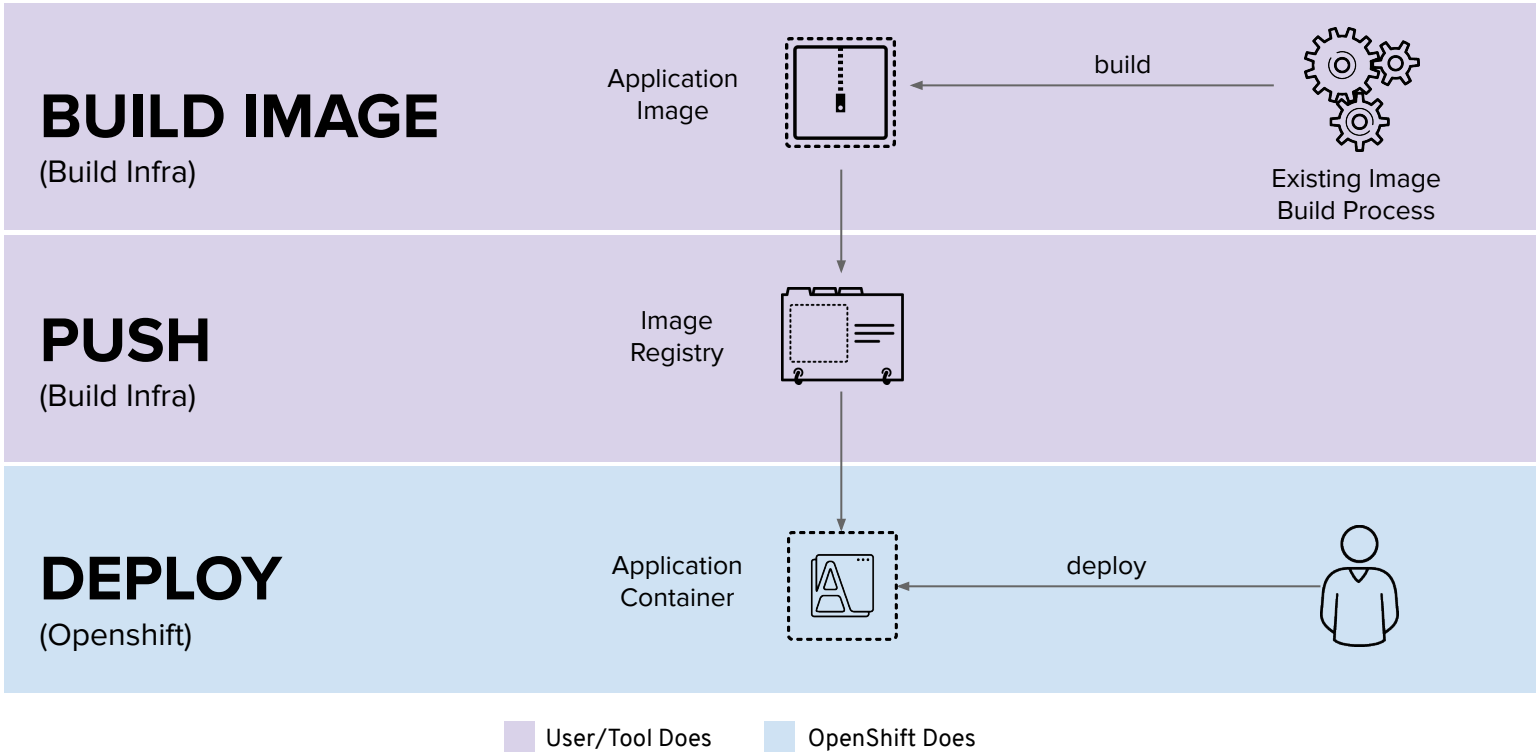
DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)



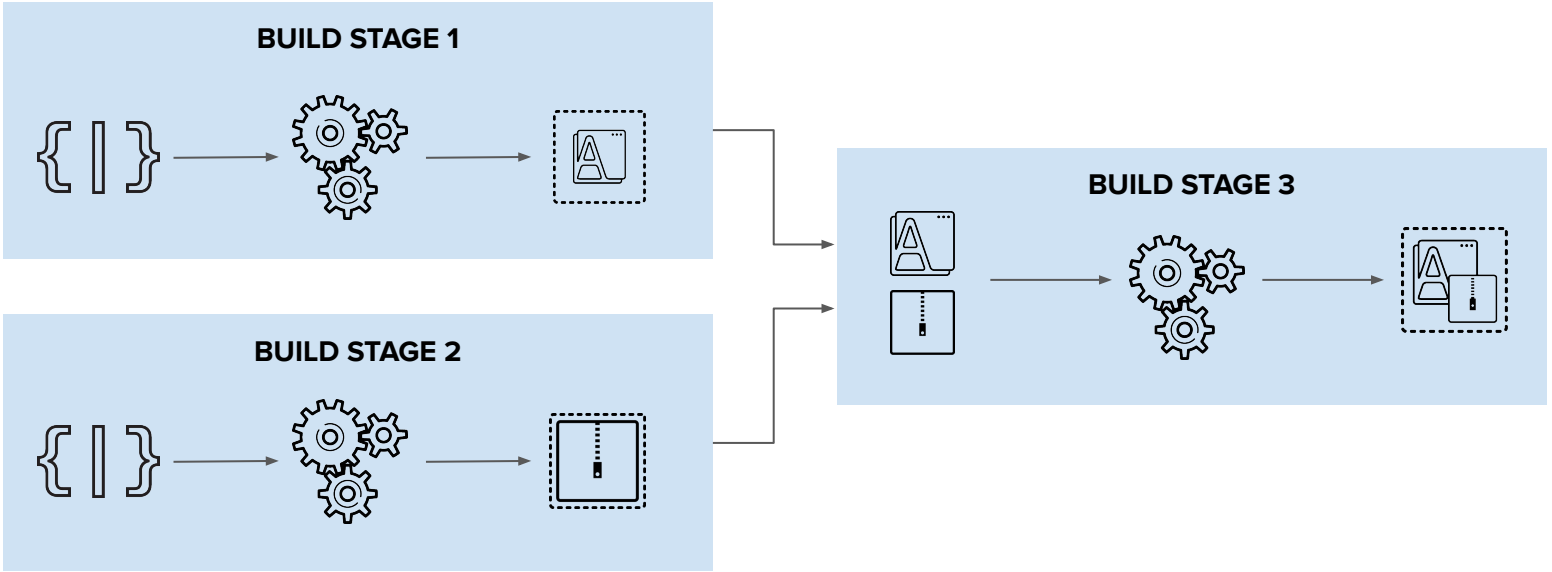
DEPLOY APP BINARY WITH SOURCE-TO-IMAGE (S2I)



DEPLOY DOCKER IMAGE

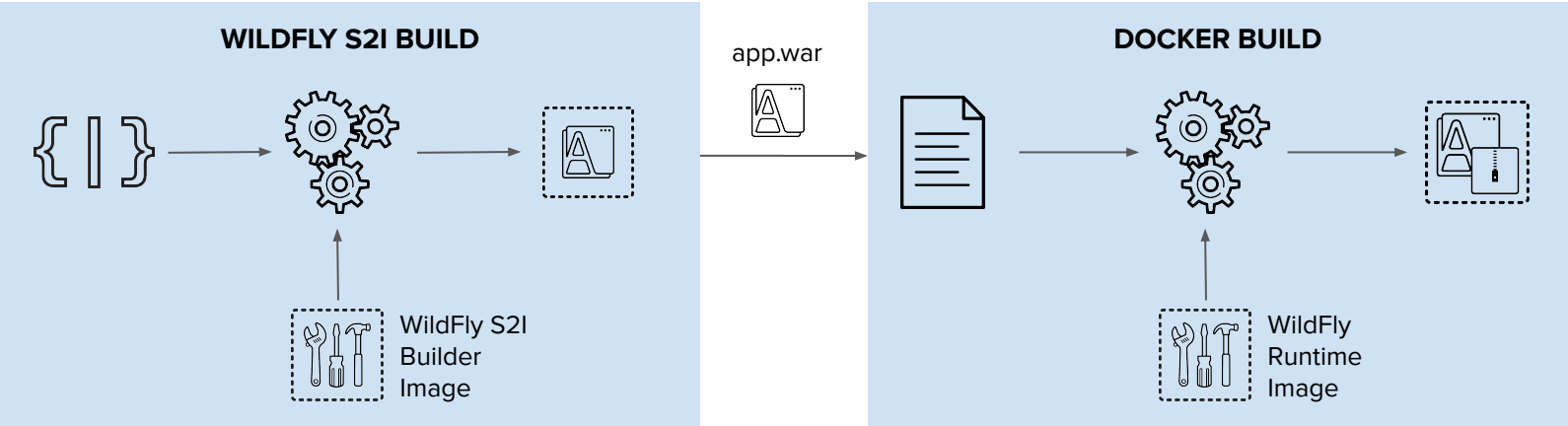


BUILD IMAGES IN MULTIPLE STAGES



EXAMPLE: USE ANY RUNTIME IMAGE WITH SOURCE-TO-IMAGE BUILDS

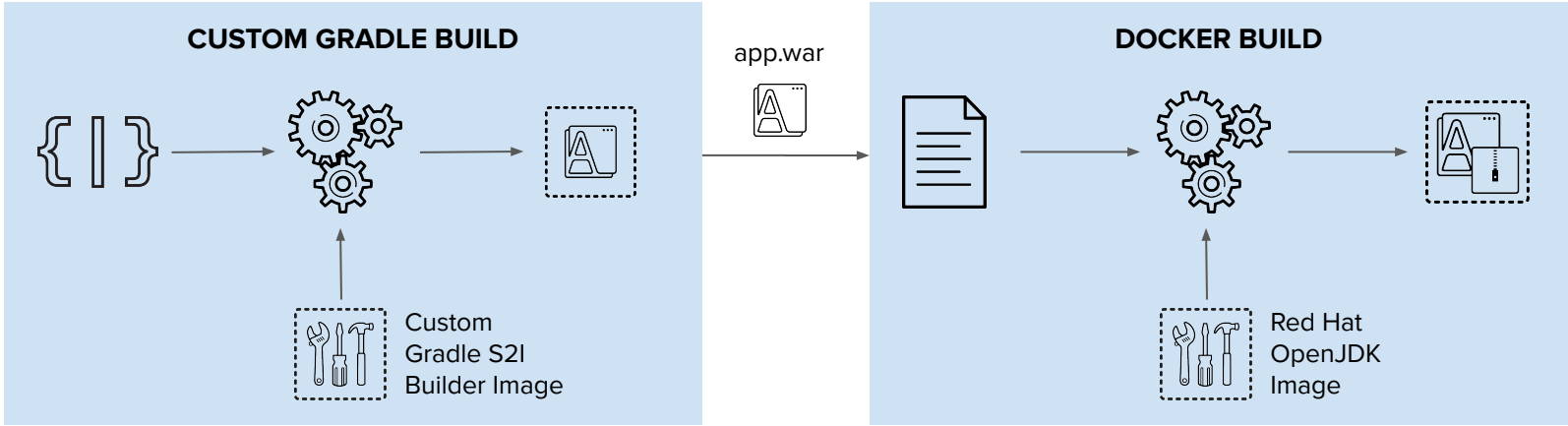
Use Source-to-Image to build app binaries and deploy on lean vanilla runtimes



read more on https://blog.openshift.com/chaining-builds/

EXAMPLE: USE ANY BUILD TOOL WITH OFFICIAL RUNTIME IMAGES

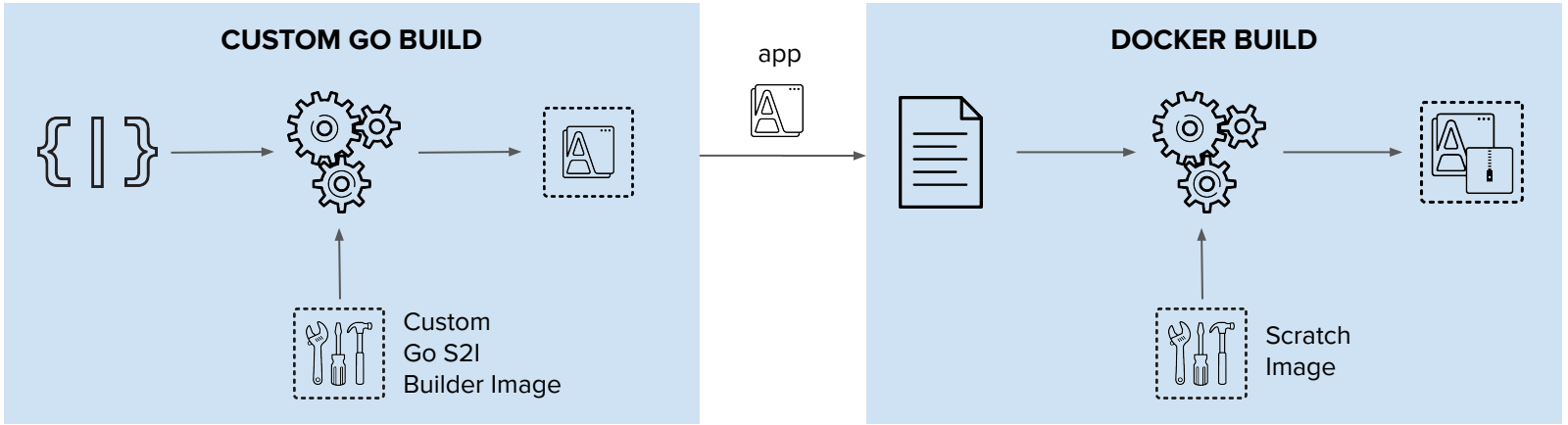
Use your choice of build tool like Gradle and deploy to official images like the JDK image



read more on https://blog.openshift.com/chaining-builds/

EXAMPLE: SMALL LEAN RUNTIMES

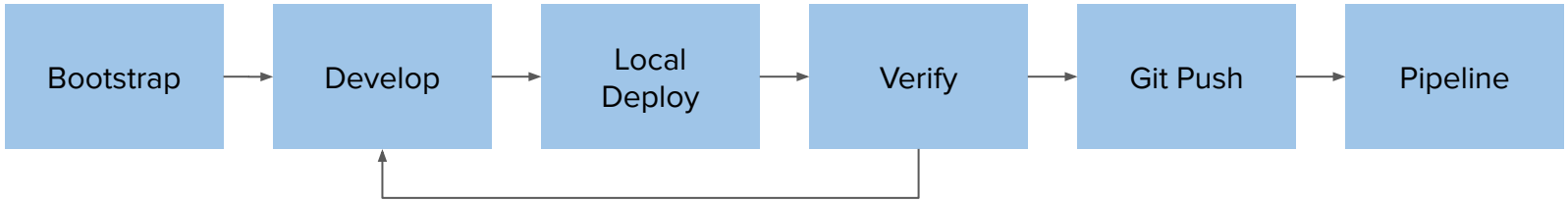
Build the app binary and deploy on small scratch images



read more on https://blog.openshift.com/chaining-builds/

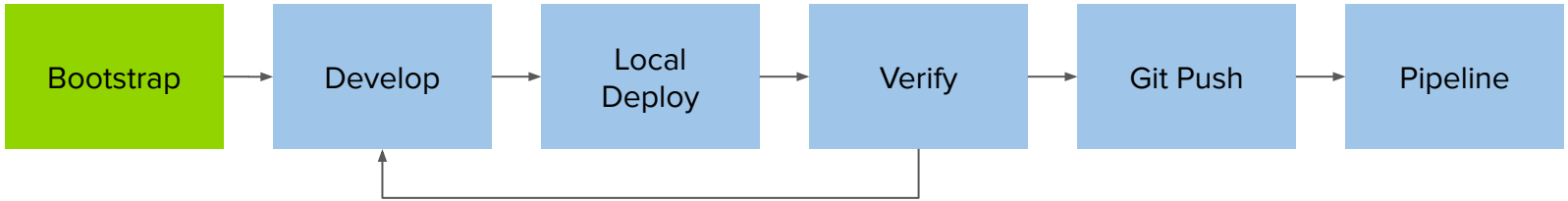
DEVELOPER WORKFLOW

LOCAL DEVELOPMENT WORKFLOW



RED HAT®
OPENSHIFT
Container Platform

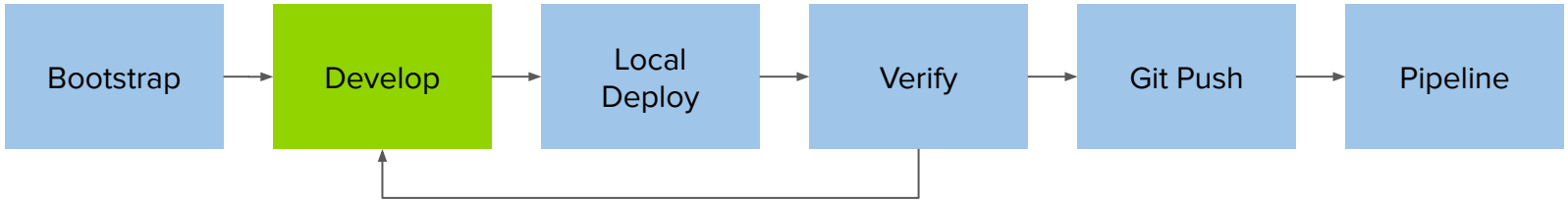
LOCAL DEVELOPMENT WORKFLOW



BOOTSTRAP

- Pick your programming language and application runtime of choice
- Create the project skeleton from scratch or use a generator such as
  - Maven archetypes
  - Quickstarts and Templates
  - OpenShift Generator
  - Spring Initializr

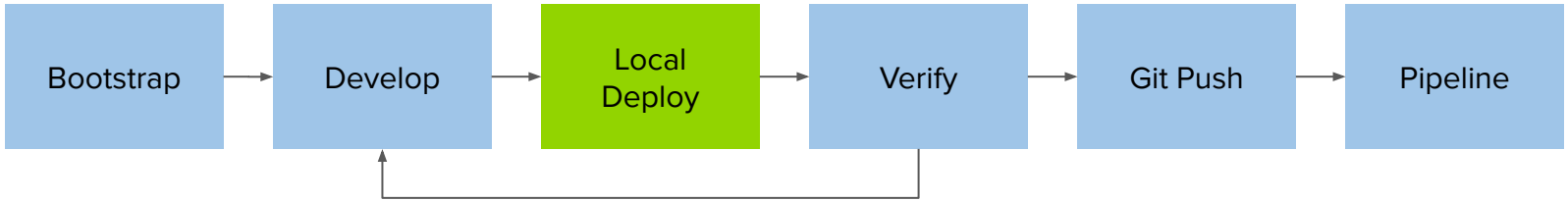
LOCAL DEVELOPMENT WORKFLOW



DEVELOP

- Pick your framework of choice such as Java EE, Spring, Ruby on Rails, Django, Express, ...
- Develop your application code using your editor or IDE of choice
- Build and test your application code locally using your build tools
- Create or generate OpenShift templates or Kubernetes objects

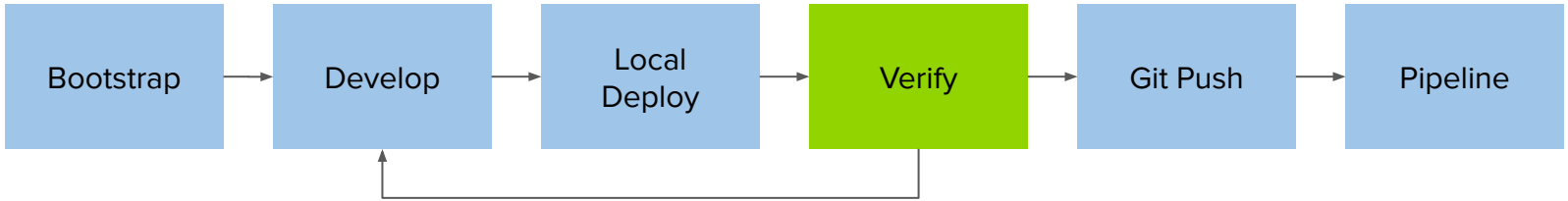
LOCAL DEVELOPMENT WORKFLOW



LOCAL DEPLOY

- Deploy your code on a local OpenShift cluster
  - Red Hat Container Development Kit (CDK), minishift and oc cluster
- Red Hat CDK provides a standard RHEL-based development environment
- Use binary deploy, maven or CLI rsync to push code or app binary directly into containers

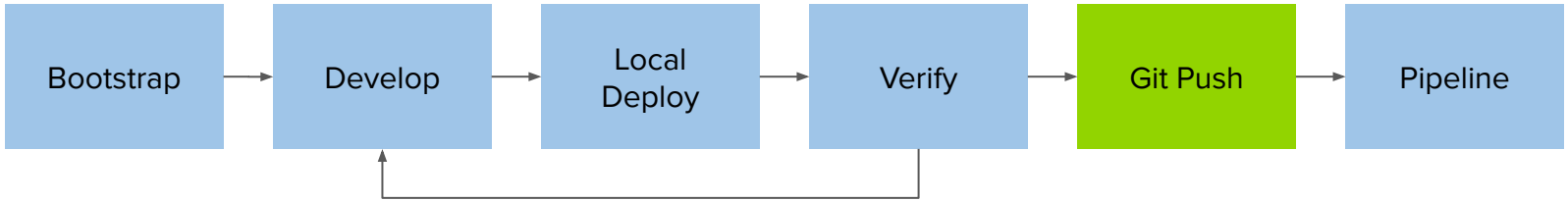
LOCAL DEVELOPMENT WORKFLOW



VERIFY

- Verify your code is working as expected
- Run any type of tests that are required with or without other components (database, etc)
- Based on the test results, change code, deploy, verify and repeat

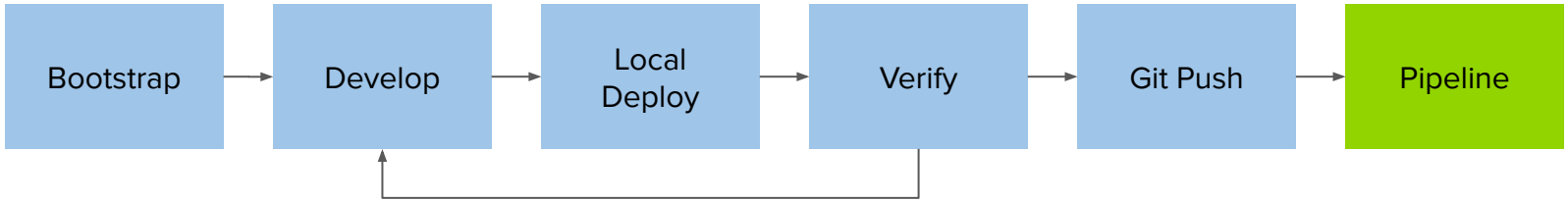
LOCAL DEVELOPMENT WORKFLOW



GIT PUSH

- Push the code and configuration to the Git repository
- If using Fork & Pull Request workflow, create a Pull Request
- If using code review workflow, participate in code review discussions

LOCAL DEVELOPMENT WORKFLOW



PIPELINE

- Pushing code to the Git repository triggers one or multiple deployment pipelines
- Design your pipelines based on your development workflow e.g. test the pull request
- Failure in the pipeline? Go back to the code and start again

Packaging Basics


JUST MOVE IT ONLINE THEY SAID
HOW HARD COULD IT BE?

What

- Now I have coded and my app works on my Kubernetes cluster
  - All fine
  - All done
- But wait…
- How to move those things from DEV to Test?
- How to release my software?
- (No, it’s not just one image)
  - Deployment / DeploymentConfig
  - Service
  - PVCs
  - ConfigMaps
  - Route


I'M SCARED NOW

What

- How to automatically recreate your App with all resources and dependencies?
- Once you've created your App with all necessary resources, you need to somehow find a way to sync it with your stages (DEV/TEST/PRE-PROD...)
- How to redistribute your App?

Zip?
Tar?
Rsynch?
Binaries?
Configuration?
Templates?
Helm Charts?
Operators?
DIY?
Kustomize?

Beer!


TOO MANY CHOICES
TOO MANY OPTIONS
imgflip.com
TROLL ME

OpenShift Templates

What

- OpenShift Templates has been the first approach in OpenShift to make your project exportable and importable
  - All the RedHat middleware products were being used via templates
  - All Samples are still coming via templates
- Helm was too basic
- Helm required Tiller (a component which required root etc.)
- Operators did not exist
- It is basically a “oc get is,bc,build,dc,is,rc,route,service -o yaml > test.yaml”
  - With some editing
  - With QUITE some editing

Can I use it?

- Short answer: Of course!
- Longer answer: If you know that you are only using OpenShift inside your own Company then please use it.
  - A Template can configure even OpenShift specific types like BuildConfigs, ImageStreams, DeploymentConfigs…
  - Easily parametrized
  - With “oc process …” there is an easy way to process a template
  - Can easily being used in your CI/CD pipelines
  - Can be versioned
  - Not too complex
  - Proven to work, easily understandable

Szenarios to use Templates

- In-Cluster movements (DEV → TEST)
- Cross-Cluster movements (TEST → PREPROD → PROD)
- Templating Application setups, including BC for use by others
- Building special examples as templates could be consumed via UI by developers
- Distributing your Apps

Drawbacks?

- Templates are OpenShift specific
- Templates are OpenShift specific
- Templates are OpenShift specific
- (And a template file could become VERY large as it contains ALL kubernetes yaml's required to run the app)

Template DEMO

kustomize.io

What

- Kustomize is a project originally founded Google
- It's in "kubectl apply -k" and "oc apply -k" now
- Has its own CLI interface, called kustomize
- It's NOT templating
- It's using overlays and patching

```
$ tree
.
├── base
│   ├── configMap.yaml
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   ├── route.yaml
│   └── service.yaml
└── overlays
    ├── production
    │   ├── deployment.yaml
    │   └── kustomization.yaml
    └── staging
        ├── kustomization.yaml
        ├── map.yaml
        └── route.yaml
```

How it works

kustomization.yaml contains information about what to do and how

```
$ cat base/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
metadata:
  name: arbitrary

# Example configuration for the webserver
# at https://github.com/monopole/hello
commonLabels:
  app: my-hello
  org: acmeCorporation

resources:
- deployment.yaml
- service.yaml
- configMap.yaml
- route.yaml
```

```
$ cat overlays/staging/kustomization.yaml
namePrefix: staging-
commonLabels:
  variant: staging
commonAnnotations:
  note: Hello, I am staging!
bases:
- ../../base
patchesStrategicMerge:
- map.yaml
- route.yaml
```

```
$ cat overlays/production/kustomization.yaml
namePrefix: production-
commonLabels:
  variant: production
commonAnnotations:
  note: Hello, I am production!
bases:
- ../../base
patchesStrategicMerge:
- deployment.yaml
- route.yaml
```

How it works

kustomize build or oc/kubectl apply -k does handle everything for you

```
$ kustomize build base
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-hello
    org: acmeCorporation
  name: the-service
spec:
  ports:
  - port: 8666
    protocol: TCP
    targetPort: 8080
  selector:
    app: my-hello
    deployment: hello
    org: acmeCorporation
  type: LoadBalancer
---
```

```
$ kustomize build overlays/staging
---
apiVersion: v1
kind: Service
metadata:
  annotations:
    note: Hello, I am staging!
  labels:
    app: my-hello
    org: acmeCorporation
    variant: staging
  name: staging-the-service
spec:
  ports:
  - port: 8666
    protocol: TCP
    targetPort: 8080
  selector:
    app: my-hello
    deployment: hello
    org: acmeCorporation
    variant: staging
  type: LoadBalancer
---
```

Can I use it?

- Short answer: Of course!
- Longer answer: If you're looking for a solution that integrates nicely with GitOps, you should definitely have a look
  - Kustomize can also configure OpenShift specific types like Routes etc.
  - It does NOT use parameters
  - With kustomize CLI there is a nice way to test your layers
  - Can easily being used in your CI/CD pipelines
  - Can be versioned
  - Not too complex
  - Proven to work, easily understandable

Scenarios to use kustomize

- In-Cluster movements (DEV → TEST)
- Cross-Cluster movements (TEST → PREPROD → PROD)
- GitOps
- OpenShift Pipelines / Tekton
- NOT usable for application publishing / distribution

Drawbacks?

- You can only change existing entries and add new ones...
- You can't use it for redistribution

Resources

Automated Application Packaging and Distribution with OpenShift - Part 1/2 – Open Sourcerers

https://kustomize.io
https://github.com/kubernetes-sigs/kustomize
https://speakerdeck.com/spesnova/introduction-to-kustomize
https://github.com/wpernath/kustomize-demo

Kustomize-DEMO