redhat.

INTRODUCTION TO JBOSS EAP 7 ON OPENSHIFT

James Falkner
Sr. Technical Marketing Manager, Middleware
April 2017

JBOSS MIDDLEWARE ON OPENSHIFT

Application Container Services

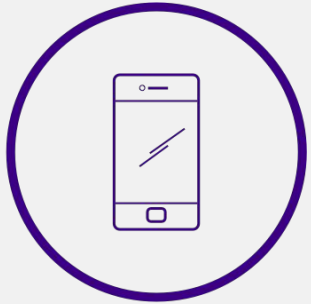
- JBoss Enterprise Application Platform
- JBoss Web Server (Tomcat)
- JBoss Developer Studio
- Red Hat SSO

Integration Services

- Fuse
- A-MQ
- Data Virtualization
- 3scale API Management

Business Process Services

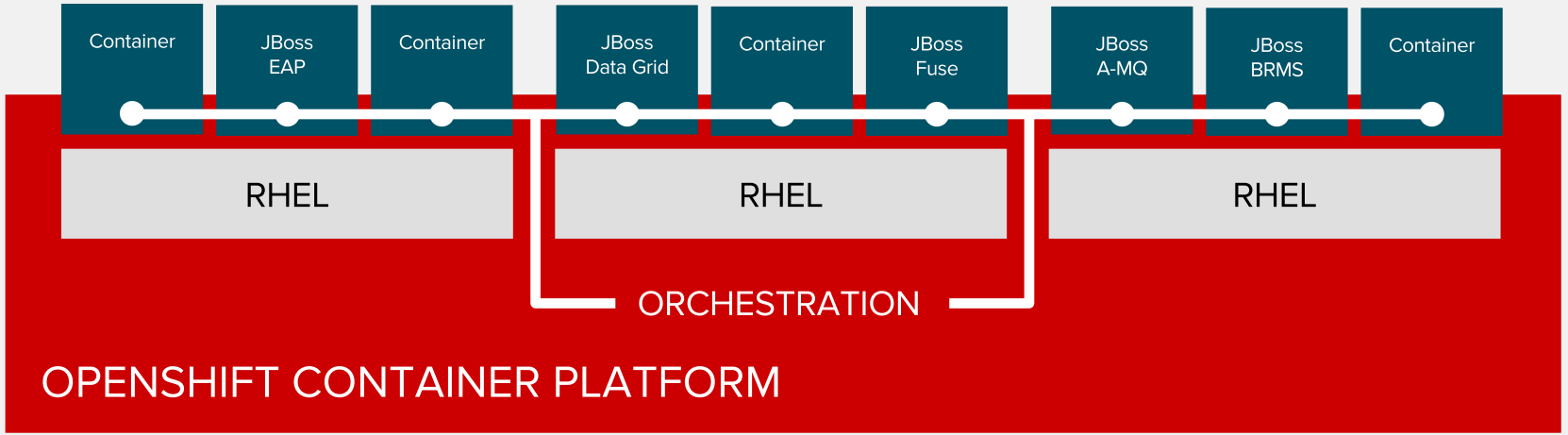
- Business Process Management
- Business Rules Management System



Mobile Services

- Red Hat Mobile / FeedHenry

RED HAT® OPENSHIFT Container Platform



JBOSS ENTERPRISE APPLICATION PLATFORM

Application Container Services

JBoss Enterprise Application Platform

JBoss Web Server / Tomcat

JBoss Data Grid

Red Hat SSO

- The most popular open source Java EE application server
- Lightweight dynamic architecture
- Choice of programming models - Java EE, Spring
- Supports full spectrum of Java workloads
- Optimized for modern IT infrastructure

WHAT ARE THE COMMON FEATURES?

- Automatic discovery, clustering, tuning
- Automatic HTTPS configuration
- S2I (Source To Image)
- Externalized, automatic app configuration
- Subscription portability
- Lightweight, containerized, fast startup
- OpenShift: scheduling, load balancing, image promotion, CI/CD pipelines, routing, RBAC, storage, image management, telemetry, quotas, multi-tenancy, ...

WHAT ARE THE DIFFERENCES?

- No web-based Management Console
- JBoss CLI is available from within the container
- No Domain mode. Not needed either when running on OpenShift!
- ActiveMQ Artemis for internal messaging. A-MQ for external
- Default ROOT page is disabled. Deploy your own application!
- Clustering is supported through Kubernetes discovery mechanisms

JBOSS EAP 7 - WHAT'S NEW?

- Java EE 7 certified
- HornetQ is replaced with Apache Artemis for messaging
- JBoss Web is replaced with Undertow
- UX Improvements
- Offline CLI
- Graceful shutdown
- Port reduction
- Developer features (Batch, websockets, quickstarts, migration tooling)
- Migration guide
  https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0.beta/migration-guide/migration-guide

LABS

MULTITENANT LAB ENVIRONMENT

- Our lab environment includes the `ovs-multitenant` plug-in which provides *project* level isolation for pods and services.
- By default, pods from different *projects* cannot send packets to or receive packets from pods and services of a different project, even if created by the same user.
- The project namespace is shared between all users of an OpenShift cluster.
- In the following labs, you must create unique project names when using `oc new-project`. Use your username, or some other unique name (but not these!). Examples:
  - `oc new-project lab1-user12`
  - `oc new-project lab5-shadowman`
- Pod names in the examples are only examples; yours will always be different and must be discovered via `oc get pods` as needed.

LABS

Lab 0 - Deploying Sonatype Nexus Maven Repository

Lab 1 - JBoss EAP Application Deployment

Lab 2 - JBoss EAP Clustering

Lab 3 - Deployment with JBoss Developer Studio

Lab 4 - Local Development with JBoss Developer Studio

Lab 5 - Creating Applications Without a Git Repository

Lab 6 - Remote Debugging

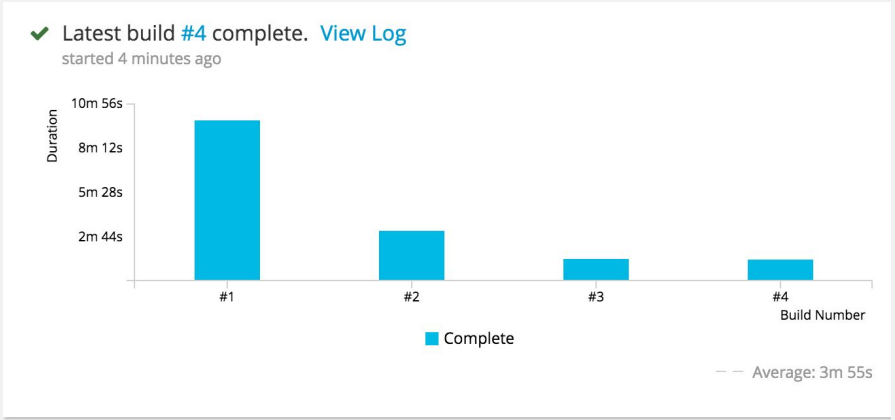
Lab 7 - Advanced JBoss EAP Configuration

LAB 0

Deploying Sonatype Nexus Maven Repository

LAB 0 - REDUCING BUILD TIME WITH NEXUS

- Using Nexus instead of public Maven repositories, causes Nexus to cache the artifacts the first time they are downloaded and removes the need to download them on every build.
- Caching artifacts reduces the build time significantly



Build	Status	Started
#4	Complete in 1 minute, 18 seconds	4 minutes ago - 5/5/16 3:31 AM
#3	Complete in 1 minute, 21 seconds	21 minutes ago - 5/5/16 3:14 AM
#2	Complete in 3 minutes, 4 seconds	26 minutes ago - 5/5/16 3:09 AM
#1	Complete in 9 minutes, 57 seconds	44 minutes ago - 5/5/16 2:51 AM

LAB 0 - DEPLOYING SONATYPE NEXUS

- A Maven repository provides a standard for storing and serving binary software
- Local repository exists in `~/.m2/repository`
- Maven Central repository: `http://repo1.maven.org/maven2/`
- Nexus is a Maven Repository Manager created by Sonatype
- Simplifies the maintenance of internal repositories and access to external repositories
- Proxies requests for external artifacts and caches the results
- Provides a deployment destination for your own generated artifacts

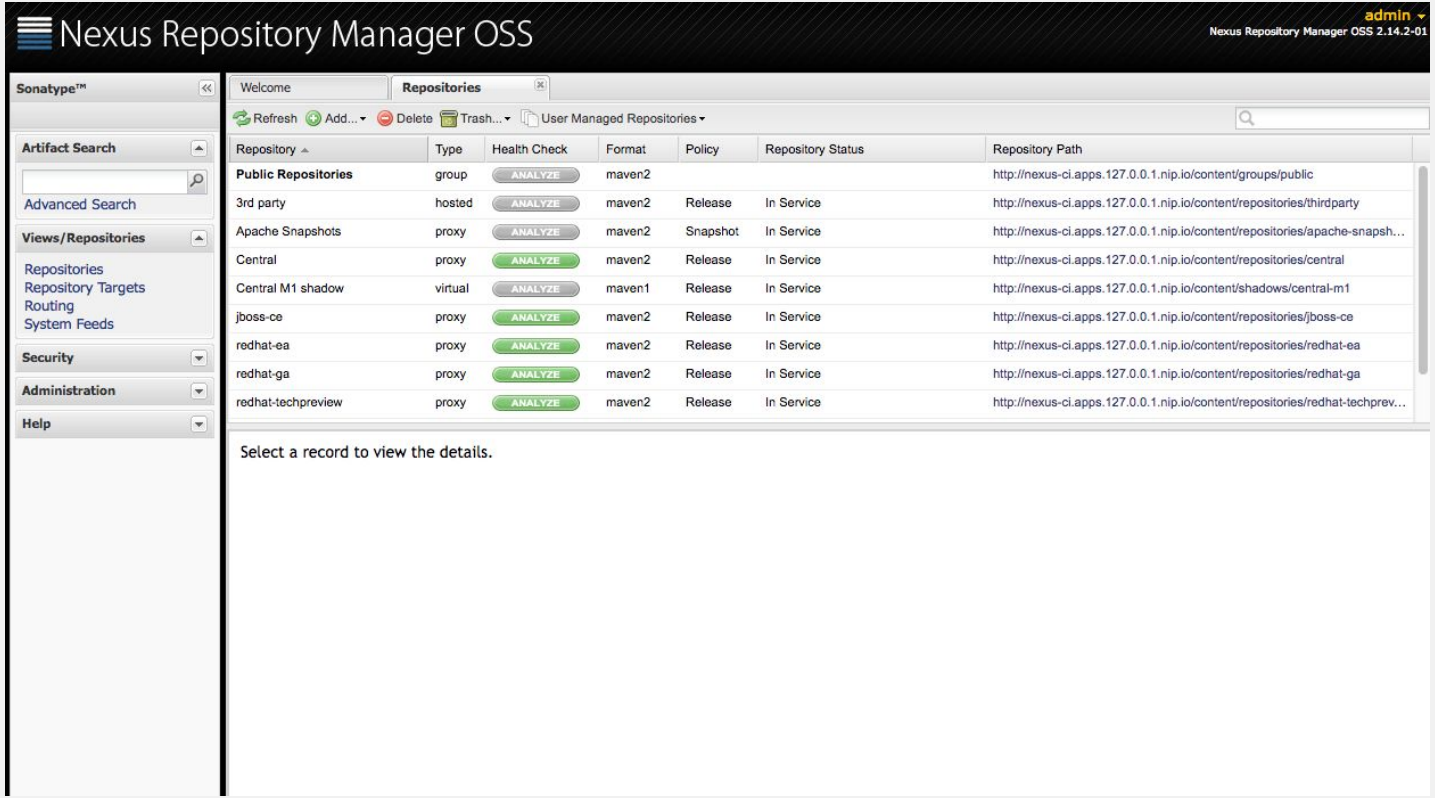
LAB 0 - DEPLOYING SONATYPE NEXUS

- Create a new project and deploy sonatype/nexus official image from Docker Hub

```
$ oc new-project ci-<username>
$ oc create -f https://raw.githubusercontent.com/OpenShiftDemos/nexus/master/nexus2-template.yaml
$ oc new-app nexus2
```

- Red Hat repositories are automatically included and configured.
- Once the deployment completes, verify Sonatype Nexus is up and running by clicking on its route link in the OpenShift web console.
- Login to Nexus with the admin user
  - Username: `admin`
  - Password: `admin123`

LAB 0 - DEPLOYING SONATYPE NEXUS



LAB 0 - REDUCING BUILD TIME WITH NEXUS

- Enabling Nexus can be achieve in the following ways
  - Setting `MAVEN_MIRROR_URL` environment variable on the `BuildConfig`
  - Provide your own maven settings with Nexus configured in `configuration/settings.xml` of your source repository.

    An example of `settings.xml` is included in Nexus docs: https://books.sonatype.com/nexus-book/reference/config-maven.html
- Create a new application using JBoss EAP S2I builder and wait for it to complete (~2-3mins)

```
$ oc new-app eap70-basic-s2i \
        --param=APPLICATION_NAME=ticket-monster \
        --param=SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/ticket-monster.git \
        --param=SOURCE_REPOSITORY_REF=2.7.0.Final \
        --param=CONTEXT_DIR=demo
```

LAB 0 - REDUCING BUILD TIME WITH NEXUS

- OpenShift has a built-in DNS so that the services can be reached by the service DNS as well as the service IP/port. Using service DNS removes the need to restart pods in order to pick up new services IP. Also removes the need to use environment variables, as the service DNS does not change.

```
<service> # when in same project
<service>.<pod_namespace> # cross project
<service>.<pod_namespace>.svc.cluster.local
```

- This DNS resolution only occurs from within the cluster network.
- Multi-tenant pod configuration prevents traffic between projects unless explicitly allowed by a cluster administrator.

LAB 0 - REDUCING BUILD TIME WITH NEXUS

- Let's rebuild the app and see how Nexus affects build times! Set `MAVEN_MIRROR_URL` on the BuildConfig to point at your previously created Maven mirror:

```
$ oc set env bc/ticket-monster MAVEN_MIRROR_URL=http://nexus:8081/content/groups/public
```

- Since the nexus service is in the same project, we simply refer to it's hostname as `nexus`
- Start a new build using this configuration and see the improved speed as Maven artifacts are cached:

```
$ oc start-build ticket-monster --follow
```

- Finally, now that all artifacts have been cached, start a final build (using the same command above) which should show a dramatic speed improvement over the first build!

LAB 1

JBoss EAP Application Deployment

LAB 1 - JBOSS EAP APP DEPLOYMENT

- In this lab, JBoss EAP 7 S2I builder is used to manage the application lifecycle
- Verify that JBoss EAP 7 Templates and ImageStream are installed

```
$ oc new-app -S eap
Templates (oc new-app --template=<template>)
-----
eap70-basic-s2i
  Project: openshift
  Application template for EAP 7 applications built using S2I.

Image streams (oc new-app --image-stream=<image-stream> [--code=<source>])
-----
jboss-eap70-openshift
  Project: openshift
  Tracks:  registry.access.redhat.com/jboss-eap-7/eap70-openshift
  Tags:    1.3, 1.3-22, 1.4, 1.4-10, latest
```

LAB 1 - JBOSS EAP APP DEPLOYMENT

- Briefly review the available documentation for containerized middleware on OpenShift:

  https://access.redhat.com/documentation/en/red-hat-jboss-middleware-for-openshift/

- Review `jboss-openshift` GitHub repository to get familiar with example templates for various xPaaS images:
  https://github.com/jboss-openshift/application-templates

- The documentation for each example template is available in the `docs` directory of the GitHub repository. For example review the docs for `eap70-basic-s2i` template:
  https://github.com/jboss-openshift/application-templates/blob/master/docs/eap/eap70-basic-s2i.adoc

LAB 1 - JBOSS EAP APP DEPLOYMENT

- Create a new project

```
$ oc new-project lab1-<username>
```

- You will deploy the **kitchensink** example from JBoss EAP 7 quickstarts https://github.com/jboss-developer/jboss-eap-quickstarts

- Examine the **eap70-basic-s2i** template and review the S2I builder config and parameters

```
$ oc get template eap70-basic-s2i -n openshift -o yaml
```

LAB 1 - JBOSS EAP APP DEPLOYMENT

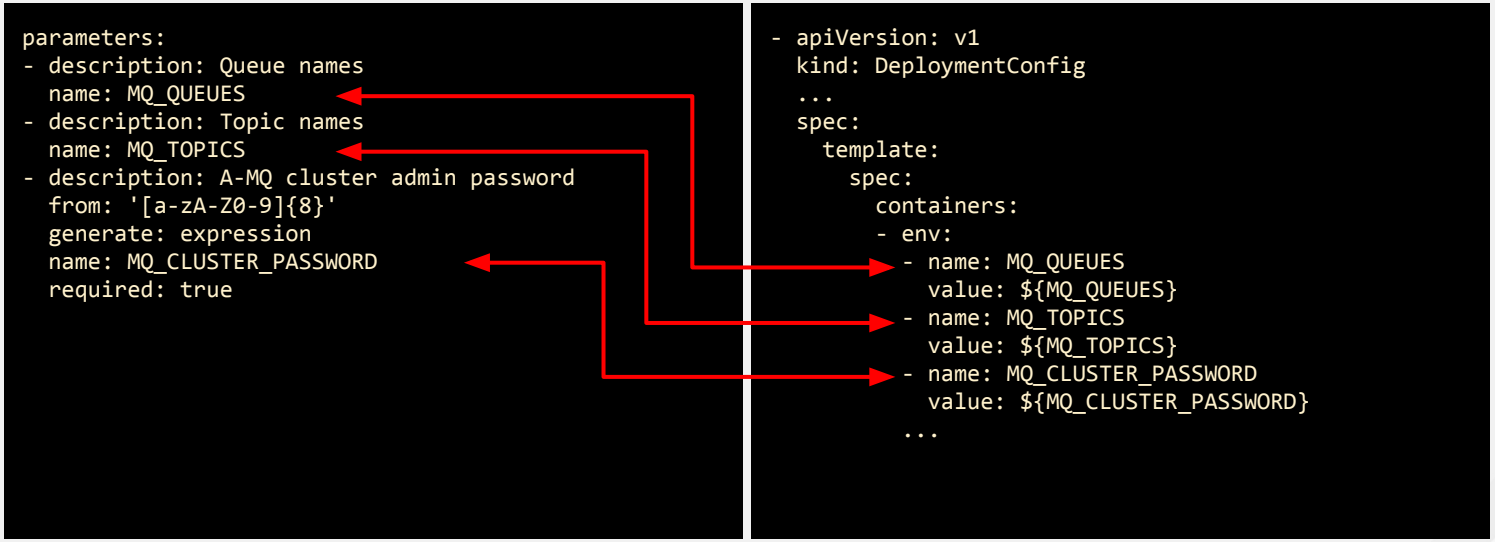
- A BuildConfig is defined in this template to build the application source code in the given Git repository and lay over the JBoss EAP 7 image

```
- apiVersion: v1
  kind: BuildConfig
  ...
  spec:
    output:
      to:
        kind: ImageStreamTag
        name: ${APPLICATION_NAME}:latest
    source:
      contextDir: ${CONTEXT_DIR}
      git:
        ref: ${SOURCE_REPOSITORY_REF}
        uri: ${SOURCE_REPOSITORY_URL}
      type: Git
    strategy:
      sourceStrategy:
        forcePull: true
```

```
        from:
          kind: ImageStreamTag
          name: jboss-eap70-openshift:1.4
          namespace: ${IMAGE_STREAM_NAMESPACE}
      type: Source
    triggers:
    - github:
        secret: ${GITHUB_WEBHOOK_SECRET}
      type: GitHub
    - generic:
        secret: ${GENERIC_WEBHOOK_SECRET}
      type: Generic
    - imageChange: {}
      type: ImageChange
    - type: ConfigChange
```

LAB 1 - JBOSS EAP APP DEPLOYMENT

- The template parameters are set on the JBoss EAP 7 container as environment variables to configure queues, datasource, etc



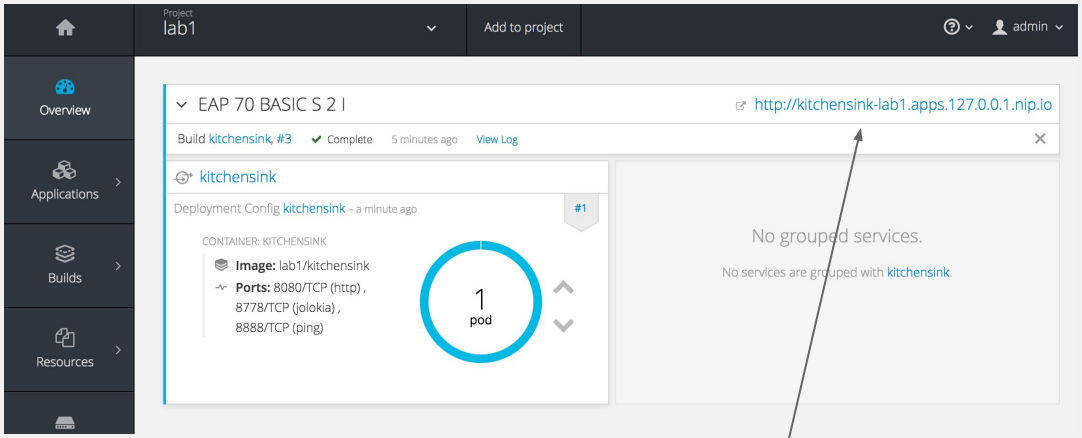
LAB 1 - JBOSS EAP APP DEPLOYMENT

- Create a new application based on the `eap70-basic-s2i` template and the `kitchensink` Git repository

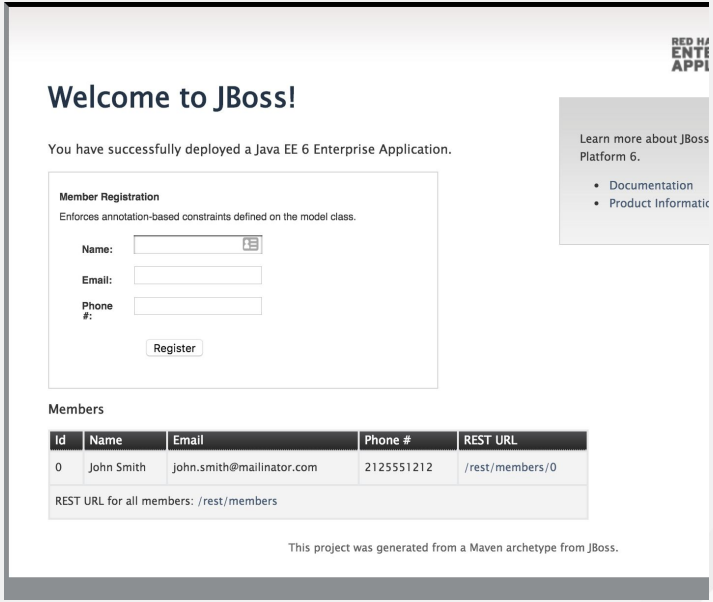
```
$ oc new-app eap70-basic-s2i \
        --param=APPLICATION_NAME=kitchensink \
        --param=SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts.git \
        --param=SOURCE_REPOSITORY_REF=7.0.0.GA \
        --param=CONTEXT_DIR=kitchensink \
        --param=MQ_QUEUES=KITCHENSINK_QUEUE
```

- Several objects are created (image stream, build config, deployment config, service, route) and the build is kicked off (you can follow with `oc logs -f bc/kitchensink` or watch the build from the OpenShift web console)
- After the build and deployment occurs (2-3 minutes), point your browser to the route url to verify the application is deployed successfully

LAB 1 - JBOSS EAP APP DEPLOYMENT



Route



LAB 1 - JBOSS EAP APP DEPLOYMENT

- Examine the `KITCHENSINK_QUEUE` queue by SSHing into the JBoss EAP 7 container using the `oc rsh` command and running JBoss CLI

```
$ oc get pods
NAME                  READY     STATUS      RESTARTS   AGE
kitchensink-1-build   0/1       Completed   0          5m
kitchensink-1-h4rc3   1/1       Running     0          29s

$ oc rsh kitchensink-1-h4rc3

sh-4.2$ $JBOSS_HOME/bin/jboss-cli.sh -c
--command="/subsystem=messaging-activemq/server=default:read-children-names(child-type=jms-queue)"

Picked up JAVA_TOOL_OPTIONS: -Duser.home=/home/jboss -Duser.name=jboss
{
    "outcome" => "success",
    "result" => [
        "DLQ",
        "ExpiryQueue",
        "KITCHENSINK_QUEUE"
    ]
}
```

LAB 1 - JBOSS EAP APP DEPLOYMENT

- The configuration variables set on `BuildConfig` and `DeploymentConfig` objects are consumed by the JBoss EAP S2I image, which contains specialized scripts that run during the build and deployment to process and set the variables in JBoss EAP configuration.
- While still rsh'd into the container, examine the S2I scripts:

```
sh-4.2$ ls -l /usr/local/s2i
total 16
-rwxr-xr-x. 1 root root 2715 Mar 17 10:07 assemble
-rwxr-xr-x. 1 root root 2331 Mar 17 10:01 common.sh
-rwxr-xr-x. 1 root root   52 Mar 17 10:07 run
-rwxr-xr-x. 1 root root   69 Mar 17 10:07 save-artifacts
sh-4.2$ cat /usr/local/s2i/assemble
sh-4.2$ cat /usr/local/s2i/run
```

- More information S2I can be found in the docs:
  - https://access.redhat.com/documentation/en-us/openshift_container_platform/3.4/html/creating_images/creating-images-s2i

LAB 1 - JBOSS EAP APP DEPLOYMENT

- OPTIONAL: The purpose of using templates is to simplify instantiation complete applications. JBoss EAP 7 S2I builder can also be used directly with `oc new-app` to build and deploy the source code on JBoss EAP 7 using the pattern:

  `[builder-image]~[git-repo]#[git-branch]`

```
$ oc new-app jboss-eap70-openshift~https://github.com/jboss-developer/jboss-eap-quickstarts.git#7.0.0.GA \
          --context-dir=kitchensink \
          --name kitchensink-direct \
          --env=MQ_QUEUES=KITCHENSINK_QUEUE
```

  OpenShift creates the required objects for building and deploying the application. Note that no routes are created this way and you need to explicitly create a route for the service.

LAB 2

JBoss EAP Clustering

LAB 2 - JBoss EAP Clustering

- Clustering is achieved through either Kubernetes or DNS discovery mechanism
- Currently KUBE_PING is the pre-configured and supported protocol

```
<subsystem xmlns="urn:jboss:domain:jgroups:4.0">
    <stack name="tcp">
        ...
        <protocol type="openshift.KUBE_PING" socket-binding="jgroups-mping"/>
        ...
</subsystem>
```

- Clustering is enabled through setting the following env vars on the container
  - `OPENSHIFT_KUBE_PING_NAMESPACE`: clustering project namespace
  - `OPENSHIFT_KUBE_PING_LABELS:` clustering labels selector
- The service account the pod is running with needs to be allowed to access Kubernetes API

LAB 2 - JBoss EAP Clustering

- Clustering is enabled on `eap70-basic-s2i` template by default

```
$ oc get template eap70-basic-s2i -o yaml -n openshift
...
- apiVersion: v1
  kind: DeploymentConfig
  ...
  spec:
    template:
      spec:
        containers:
        - env:
          - name: OPENSHIFT_KUBE_PING_LABELS
            value: application=${APPLICATION_NAME}
          - name: OPENSHIFT_KUBE_PING_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          ...
```

LAB 2 - JBoss EAP Clustering

- Create a new project

```
$ oc new-project lab2-<username>
```

- Authorize the default service account to be able to access Kubernetes API. If you are using other service accounts, authorize that instead.

```
$ oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project -q)
```

- Create a JBoss EAP application based on `eap70-basic-s2i`

```
$ oc new-app eap70-basic-s2i \
     --param=APPLICATION_NAME=clustered-helloworld \
     --param=SOURCE_REPOSITORY_URL=https://github.com/siamaksade/openshift-builds.git \
     --param=SOURCE_REPOSITORY_REF=master \
     --param=CONTEXT_DIR=clustered-helloworld
```

LAB 2 - JBoss EAP Clustering

- Scale the application to 3 pods

```
$ oc scale dc clustered-helloworld --replicas=3
```

- Verify in the pods logs that clustering is enabled and service account has sufficient access to Kubernetes API (it may take a minute or two for all builds to complete and pods to start)

```
$ oc get pods | grep Running
clustered-helloworld-1-b72w9   1/1       Running     0          1m
clustered-helloworld-1-gh54v   1/1       Running     0          1m
clustered-helloworld-1-w3ho0   1/1       Running     0          1m


$ oc logs clustered-helloworld-1-b72w9 | head
Service account has sufficient permissions to view pods in kubernetes (HTTP 200). Clustering will be available.
...
```

LAB 2 - JBoss EAP Clustering

- Verify in the pod logs that a cluster is formed with 3 members

```
$ oc logs clustered-helloworld-1-b72w9 | grep view

12:17:37,123 INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-5,ee,ered-helloworld-1-clk7l) ISPN000094: Received new cluster view for channel hibernate:
[ered-helloworld-1-clk7l|2] (3) [ered-helloworld-1-clk7l, ered-helloworld-1-zdgc3, ered-helloworld-1-poiah]
```

DEVELOPMENT ENVIRONMENT

LAB 3

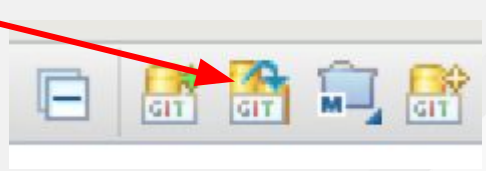
Deployment with JBoss Developer Studio

LAB 3 - DEPLOYMENT WITH JBDS

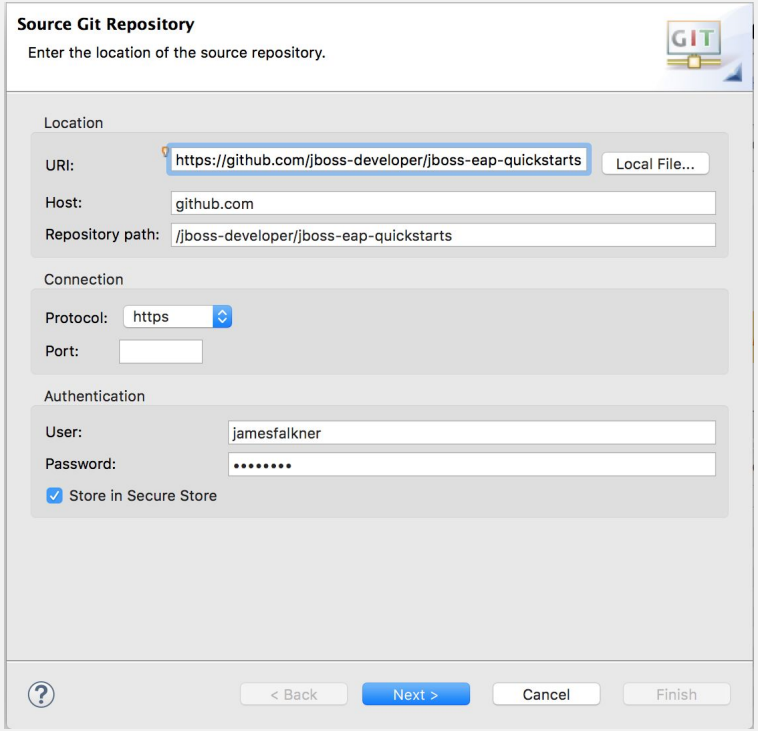
- JBDS OpenShift plugins integrate with OpenShift in order to manage application directly from IDE and reducing the developer round-trip time. In this lab, you will clone a Git repo and deploy to OpenShift through JBoss Developer Studio
- Start with importing the `jboss-eap-quickstarts` repo into your JBDS workspace
  https://github.com/jboss-developer/jboss-eap-quickstarts
- Switch to Git perspective

  Window ➜ Perspective ➜ Open Perspective ➜ Other ➜ Git ➜ OK

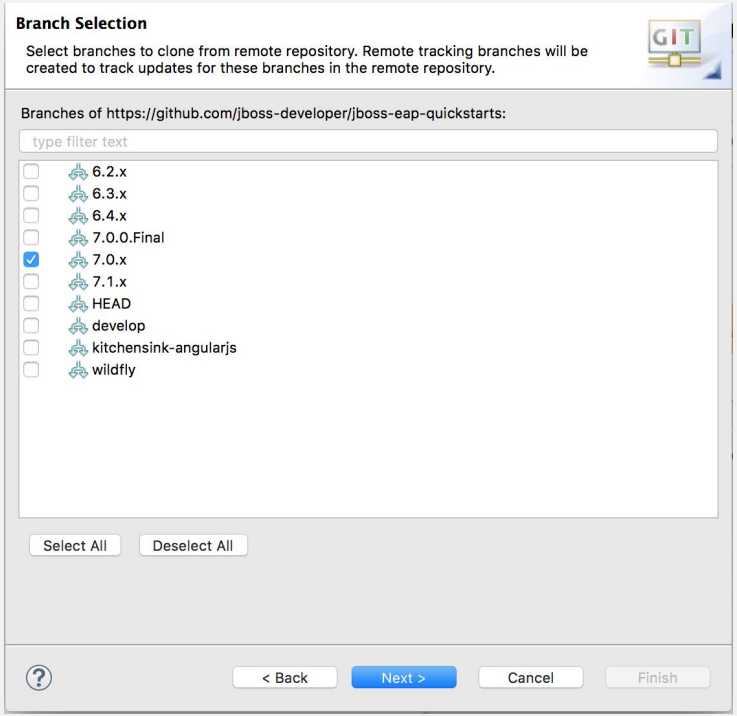
- Click on *Clone a Git Repository* in the Git Repositories view
- Specify the above Git repo as the URI and click *Next*
- Make sure only branch **7.0.x** is selected
- Specify the local clone location (or use defalt) and click *Finish*

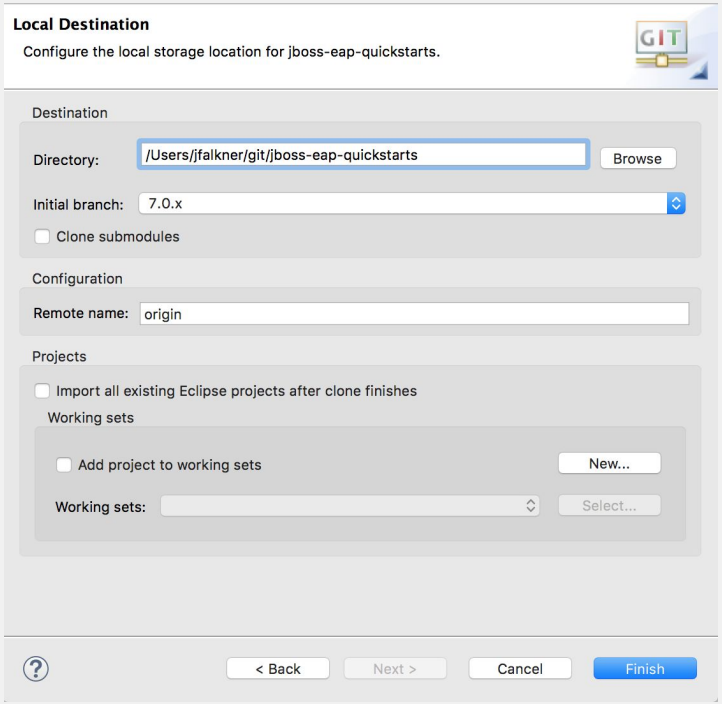


LAB 3 - DEPLOYMENT WITH JBDS



LAB 3 - DEPLOYMENT WITH JBDS

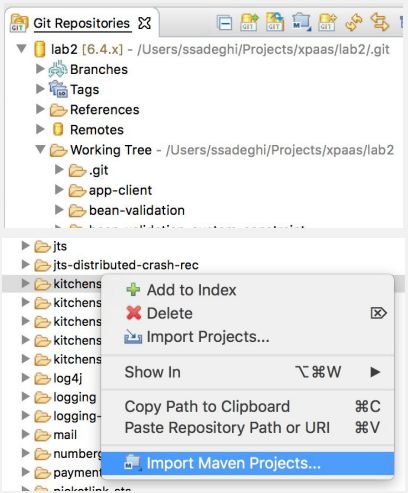




LAB 3 - DEPLOYMENT WITH JBDS

- In the Git Repositories view, click on *Working Tree*, find the `kitchensink` project and right-click on it and choose *Import Maven Projects* and then *Finish*
- If you get a popup about Cheatsheets, click No/Cancel
- Switch to the JBoss perspective by clicking the JBoss logo in the upper-right area:

- The `kitchensink` Maven project is available in the Project Explorer view

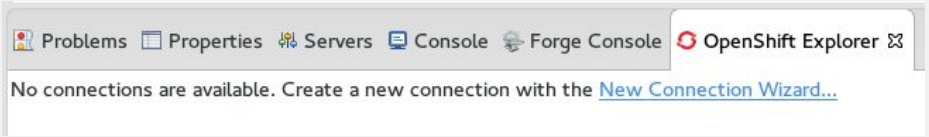


LAB 3 - DEPLOYMENT WITH JBDS

- Open OpenShift Explorer view

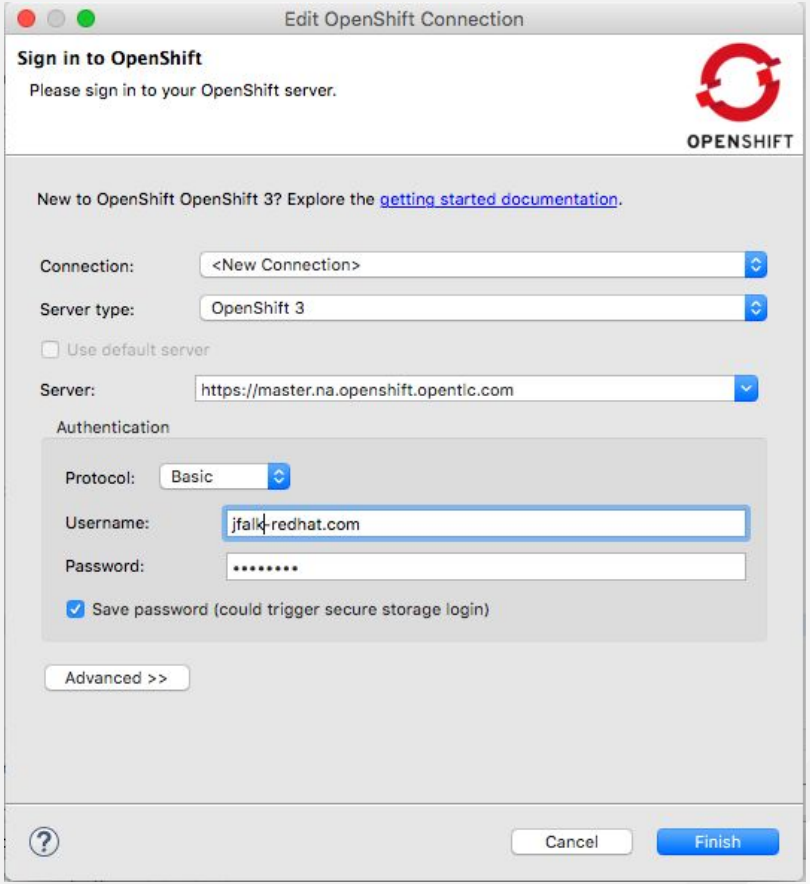
  Select Window ➜ Show View ➜ Other ➜ JBoss Tools ➜ OpenShift Explorer ➜ OK

- Create a new connection by clicking the New Connection Wizard link



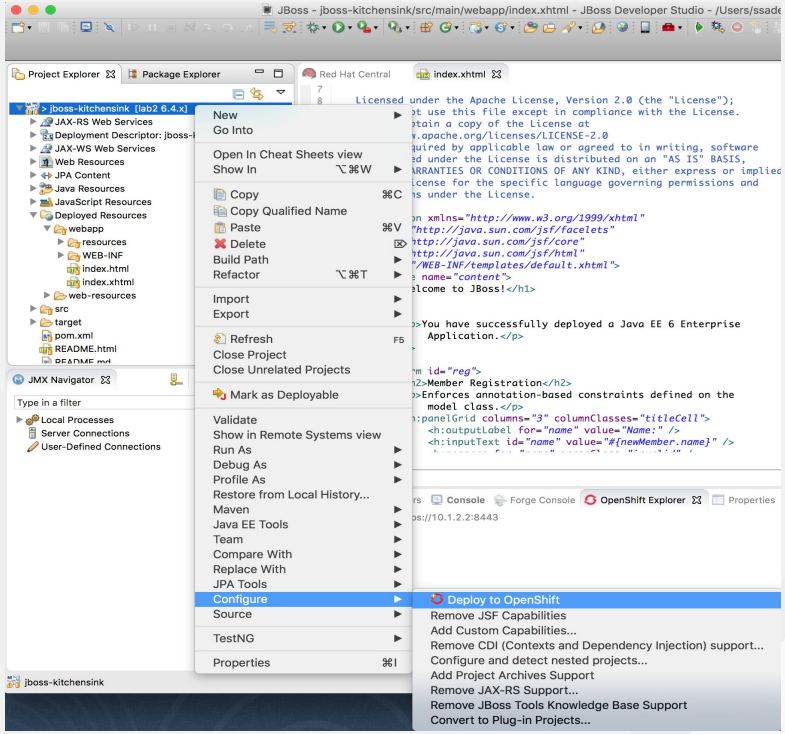
LAB 3 - OpenShift Tooling on JBDS

- Specify the OpenShift connection info
  - Connection: <New Connection>
  - Server Type: OpenShift 3
  - Use default Server checkbox: NOT checked
  - Server: https://<OCP MASTER>
  - Authentication Protocol: Basic
  - Username: <your OCP username>
  - Password: <your OCP password>
- Accept any security cert warnings
- A new OpenShift server is available in the Explorer



LAB 3 - DEPLOYMENT WITH JBDS

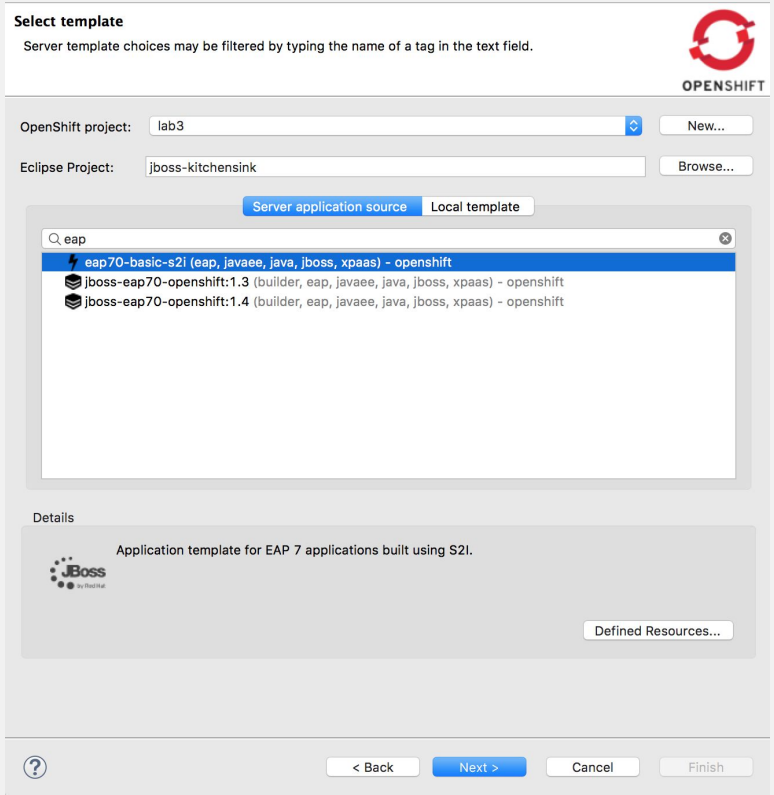
- Right click on the `jboss-kitchensink` project in Project Explorer and click on *Configure* ➔ *Deploy to OpenShift*
- If presented with an OpenShift login, re-type your username/password and ensure *Basic* auth protocol is selected and click *Next*.

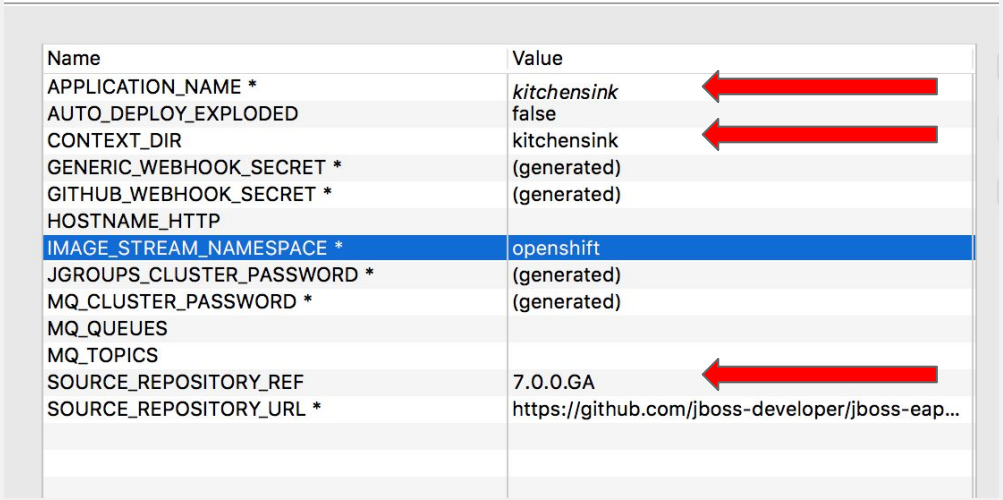


LAB 3 - DEPLOYMENT WITH JBDS

- Click `New…` next to *OpenShift Project* and enter Project Name `lab3-<username>` (optionally giving it a descriptive Display Name and Description) and click *OK*.
- Ensure your new project is now selected in the *OpenShift project* drop-down.
- Choose `eap70-basic-s2i` from the Server Templates. If you create a project specific template (in practice you should, but not today!), you can pick that from the Local Template window.
- Click on Defined Resources to review the resources that will be created by the template, then click OK to dismiss.
- Click Next and review the template parameters
    - Change the `APPLICATION_NAME` parameter to `kitchensink`
    - Change the `CONTEXT_DIR` parameter to `kitchensink`
    - Change the `SOURCE_REPOSITORY_REF` parameter to `7.0.0.GA`
- Click Next and review the labels
- Click *Finish*, review what was created in OpenShift and click and *OK*
- Examine the OpenShift Explorer view for the newly created application

LAB 3 - DEPLOYMENT WITH JBDS

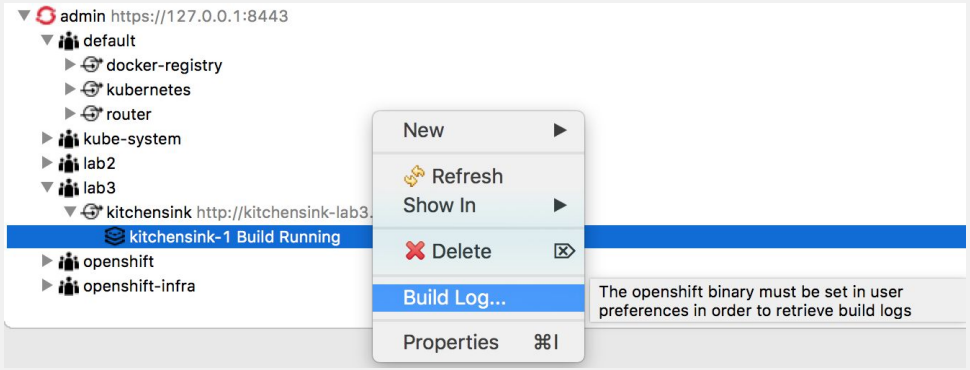


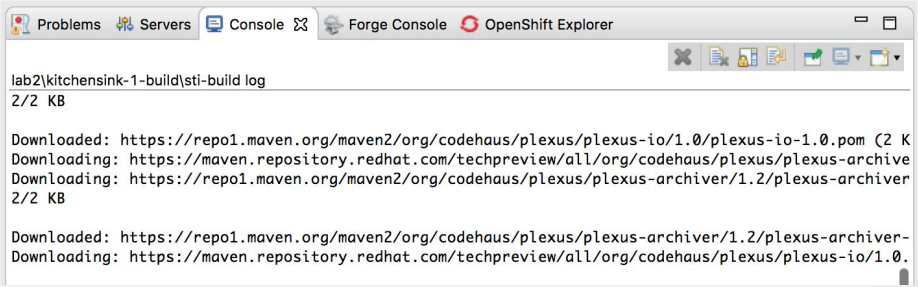


redhat.

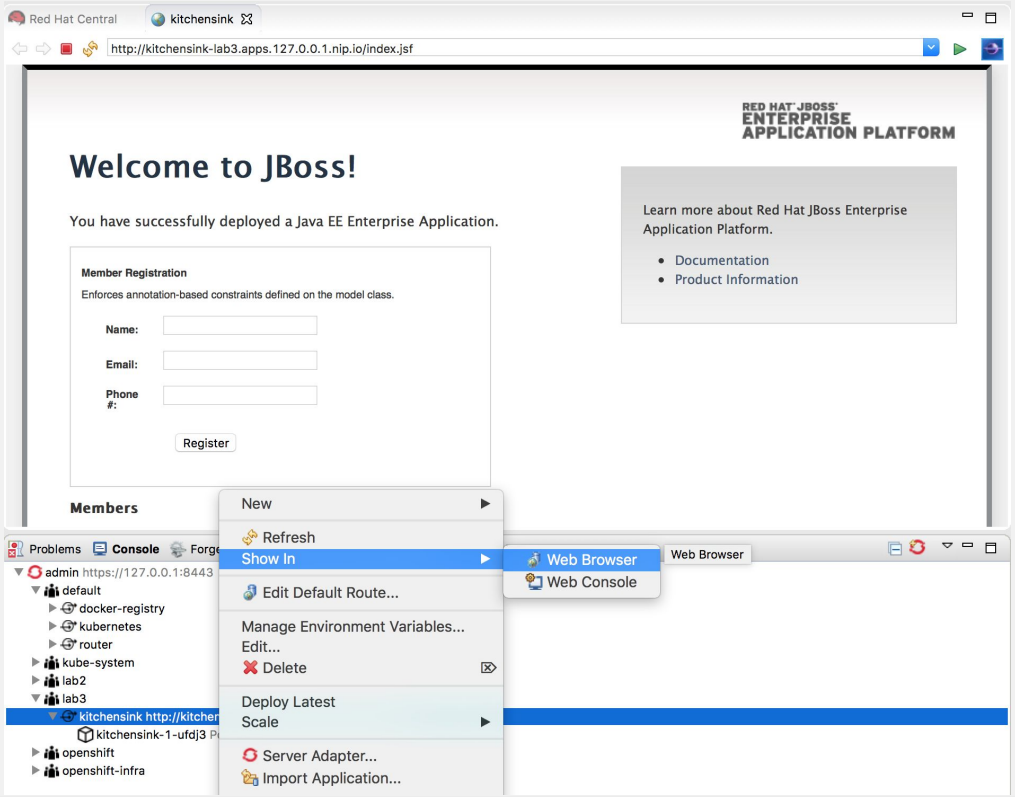
LAB 3 - DEPLOYMENT WITH JBDS

- Right click on the builder container to click on Build Log to view the build logs in the Console
- After deployment is complete, right click on the Pod and click on `Pod  Log` to view the Pod logs in the Console
- Right click on `kitchensink` in OpenShift Explorer and choose *Show In -> Web Browser* to verify app is running successfully





LAB 3 - DEPLOYMENT WITH JBDS



LOCAL DEVELOPMENT

- During development, not all changes to the code will be committed to the Git repository. Developers often modify code and want to try it immediately skipping Git commit and even Maven builds to increase productivity.
- The `oc rsync` command allows copying local files to or from a remote container
- JBoss Developer Studio takes advantage of `oc rsync` command to seamlessly copy code changes from the IDE to a remote container
- The developer workflow in that case would be

  Change Code ➜ Save ➜ Automatically Update Running Container
  Change Code ➜ Save ➜ Manual Code Publish ➜ Update Running Container

- Note that Git commit and Maven build is not part of the workflow when not needed

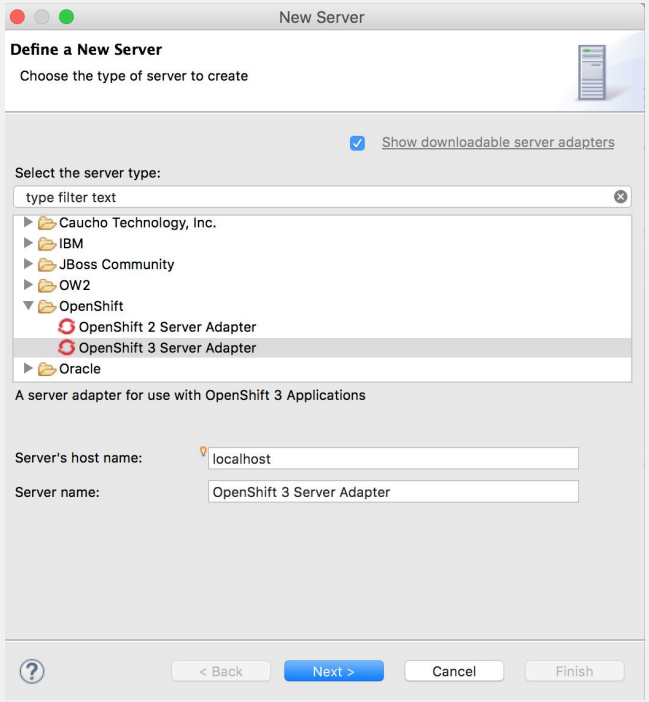
LAB 4

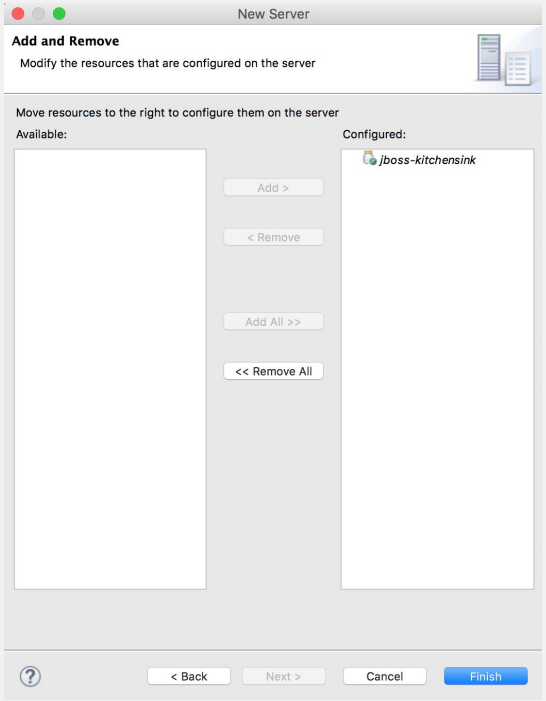
Local Development with JBDS

LAB 4 - LOCAL DEVELOPMENT WITH JBDS

- Right click in the Servers view and click on New → Server
- Choose OpenShift 3 Server Adapter
- Ensure the hostname is `localhost` and click Next
- OpenShift connection that was created previously is selected by default. Click Next to login to OpenShift
- Choose `lab3-<username>` → `kitchensink` in the Services panel and click Next
- Add `jboss-kitchensink` project to the Configured list (by selecting it and clicking *Add >*) and click Finish
- Explore `jboss-kitchensink` in the Servers view, under OpenShift 3 server

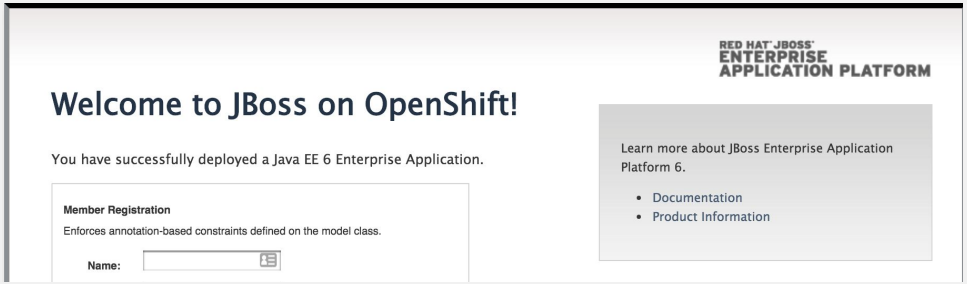
LAB 4 - LOCAL DEVELOPMENT WITH JBDS





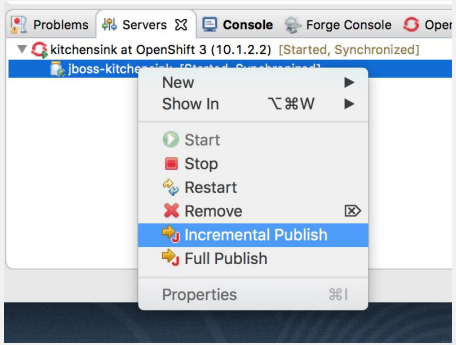
LAB 4 - LOCAL DEVELOPMENT WITH JBDS

- In the Project Explorer, open *Deployed Resources* ➜ *webapp* ➜ *index.xhtml*
- This is the home page of kitchensink application. Look for the header “Welcome to JBoss” and changed that to “Welcome to JBoss on OpenShift”
- Save the file (CTRL-S or CMD-S)
- Point your browser at the route url for `kitchensink` (or right-click on the kitchensink service in the OpenShift Explorer) and verify that the change is immediately applied to the deployed application in the container.



LAB 4 - LOCAL DEVELOPMENT WITH JBDS

- In order to disable auto-publishing the code changes, double-click on the `kitchensink at OpenShift 3` in the Servers view to open its configuration screen
- Expand the Publishing area and change to *Never Publish Automatically* and Save the file (CTRL-S / CMD-S)
- Modify index.xhtml and change the header to “Welcome to JBoss EAP on OpenShift”
- Check the deployed application in the browser. The container is not updated.
- In the Servers view, right click on `jboss-kitchensink` and click on Incremental Publish
- Check the deployed application in the browser again. The container is updated.



LOCAL DEVELOPMENT

- Deploying applications on OpenShift through S2I requires a Git repository
- Not all projects start with a Git repository
- OpenShift allows overriding the Git repository with a local directory during the build, for example:

```
$ oc start-build --from-dir=<dir_name>
```

LAB 5

Creating Applications Without Git Repository

LAB 5 - CREATING APPS WITHOUT GIT REPO

- Download the source code for the sample NodeJS application and unpack the archive

```
$ cd ~
$ curl -sL https://github.com/jboss-developer/ticket-monster/archive/2.7.0.Final.tar.gz | tar xvzf -
```

- Create a new project and then a new application based on JBoss EAP 7 S2I builder with any Git repository. You can use any Git repository since it will be overridden during the build by a local directory

```
$ oc new-project lab5-<username>
$ oc new-app jboss-eap70-openshift~http://nogit.git --name ticket-monster

--> Creating resources with label app=ticket-monster ...
    ImageStream "ticket-monster" created
    BuildConfig "ticket-monster" created
    DeploymentConfig "ticket-monster" created
    Service "ticket-monster" created
--> Success
```

LAB 5 - CREATING APPS WITHOUT GIT REPO

- The build fails due to an invalid Git repository however this is expected! You can cancel the build using oc cancel-build command to skip waiting for the build to fail.

```
$ oc cancel-build ticket-monster-1
```

- Start a new build with `oc start-build` command and point to source directory for the sample application

```
$ cd
$ oc start-build ticket-monster --from-dir=./ticket-monster-2.7.0.Final/demo --follow
```

- The source directory gets uploaded to the container and build gets started to package and deploy the source code

DEBUGGING APPLICATIONS

REMOTE DEBUG

- Java Debug Wire Protocol (JDWP) is used for communication between debugger and JVM
- JDWP lets the debugger connect remotely and step through source code as it executes
- Remote debug is enabled through passing JDWP options to JVM

```
-agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n
```

- JBoss Developer Studio can connect to JBoss EAP application and trace code execution line by line

LAB 6

Remote Debug

LAB 6 - REMOTE DEBUG

- JDWP configuration in JBoss EAP is specified in `$JBOSS_HOME/bin/standalone.conf` and commented out by default.

```
#JAVA_OPTS="$JAVA_OPTS -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"
```

- You can enable JDWP in a JBoss EAP xPaaS container by setting the `DEBUG` environment variable to `true` on the container

LAB 6 - REMOTE DEBUG

- Modify the `DeploymentConfig` for `kitchensink` application in project `lab3` and add an `DEBUG=true` env var and check that it is set:

```
$ oc project lab3-<username>
$ oc set env dc/kitchensink DEBUG=true

$ oc env dc/kitchensink --list
# deploymentconfigs kitchensink, container kitchensink
OPENSHIFT_KUBE_PING_LABELS=application=kitchensink
# OPENSHIFT_KUBE_PING_NAMESPACE from field path metadata.namespace
MQ_CLUSTER_PASSWORD=g6ukKFRS
MQ_QUEUES=
MQ_TOPICS=
JGROUPS_CLUSTER_PASSWORD=xBqEaSkQ
AUTO_DEPLOY_EXPLODED=false
DEBUG=true
```

- This will also automatically trigger a new deployment (but not a new build!)
- JVM listens by default on port 8787 as configured by JBoss EAP. You can change the default port via setting `DEBUG_PORT` environment variable

LAB 6 - REMOTE DEBUG

- Since the `DeploymentConfig` has a `ConfigChange` trigger defined, the edit will trigger a new deployment
- Review JBoss EAP logs and notice the log entry for JVM listening on port 8787

```
$ oc get pods
$ oc logs kitchensink-2-m34fd | grep -i listening
...
Listening for transport dt_socket at address: 8787
...
```

- Port 8787 is not exposed by the JBoss EAP container. In order to make the port accessible, use `oc port-forward` command to create a network tunnel between TCP port 8787 of JBoss EAP pod and local TCP port on local workstation

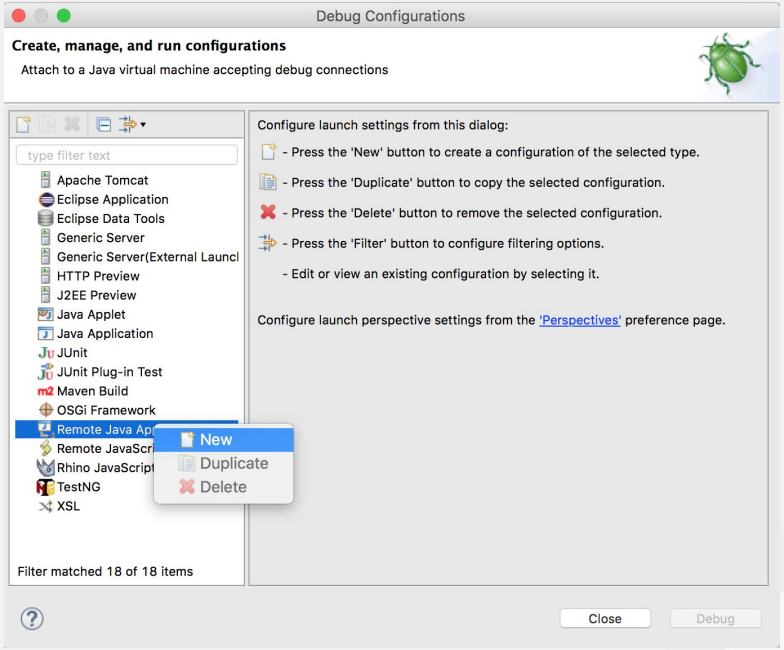
```
$ oc port-forward kitchensink-4-9wh27 8787

I0501 23:26:36.237611   48687 portforward.go:213] Forwarding from 127.0.0.1:8787 -> 8787
I0501 23:26:36.237685   48687 portforward.go:213] Forwarding from [::1]:8787 -> 8787
```

LAB 6 - REMOTE DEBUG

- Start JBoss Developer Studio
- If `kitchensink` project is not available in Project Explorer, follow Lab 3 to import it
- Right-click on the `jboss-kitchensink` project and click on

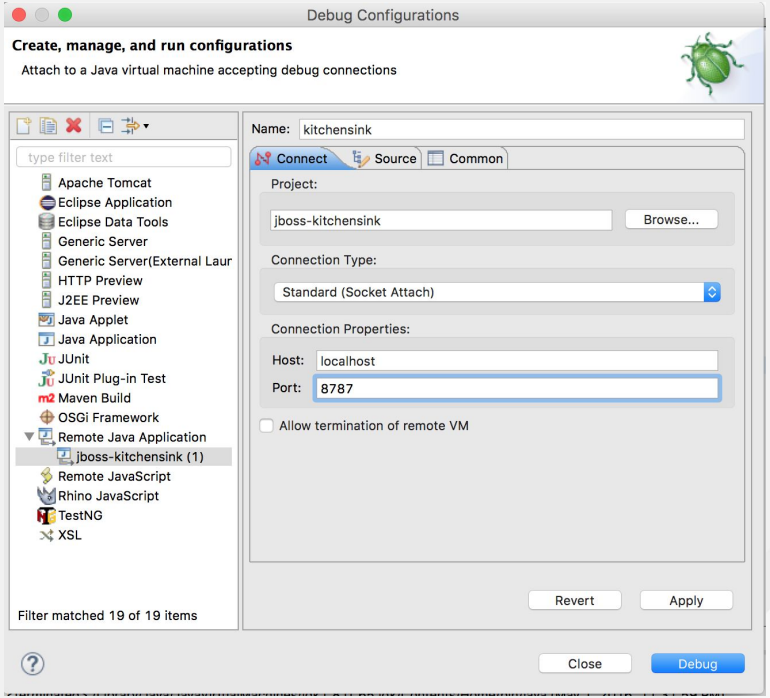
  Debug As ➔ Debug Configurations

- In the left side-bar of Debug Configuration window, right-click on *Remote Java Application* and select *New*



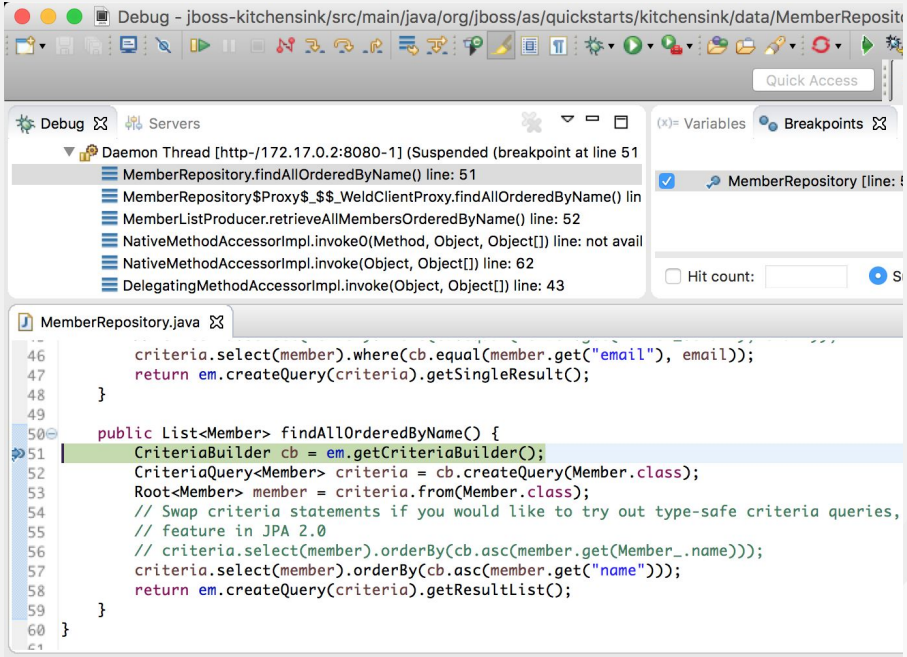
LAB 6 - REMOTE DEBUG

- Enter 8787 for Port and click on Debug
- JBDS connects to JBoss EAP in the container and the remote debug session gets started (if you get a message about Errors in the project, simply click *Proceed*)
- Open `MemberRepository.java` (press Ctrl/Cmd + Shift + R and type the class name)
- Set a breakpoint (Ctrl/Cmd + Shift + B) inside the `findAllOrderedByName()` method



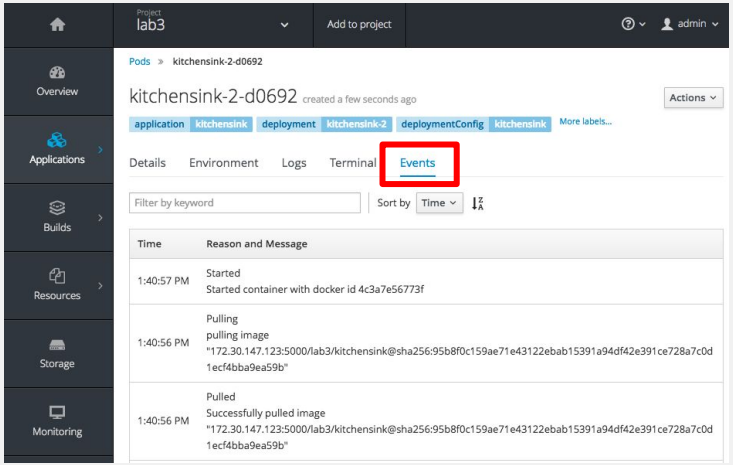
LAB 6 - REMOTE DEBUG

- Point your browser at the route url for `kitchensink`
- Notice JBDS switches to Debug Perspective and stops at the breakpoint
- You can now control the code execution via Step Over, Step Into, etc as you would do when debugging a local application



PROJECT EVENTS

- Events associated with project help understanding what applications are doing e.g. when a build or deployment fails
- Viewing events is done through OpenShift Web Console or `oc get events` command



MASTER AND NODE LOGS

- Master and Node logs are helpful for debugging administrative issues
- Access to logs is provided via SSH to the hosts (requires shell access!)
  - Reading `/var/log/messages`
  - Using `journalctl` as `root`
- Journal is used on RHEL 7 for accessing logs

```
$ journalctl                         # view logs
$ journalctl -n 10                   # view last 10 recent log entries
$ journalctl -o verbose              # view full metadata for log entries
$ journalctl -f -u openshift-node    # view only openshift node logs
```

- Adjusting Master log level

```
$ sudo vi /etc/sysconfig/openshift-master      # edit OPTIONS=--loglevel=4
$ sudo -i systemctl restart openshift-master.service
```

JBOSS EAP LOGS

- Console (Pod) log is available via OpenShift Web Console and `oc logs` command
- Does not display other handlers e.g. file
- JBoss EAP logs are accessible on the Pod

OPENSHIFT CLI LOGS

- OpenShift CLI commands call the OpenShift Master API behind the scene
- CLI logs provide details regarding the exact network communication that occurs between the `oc` client on your local workstation and the OpenShift Master API
- Change `oc` log level through `--loglevel` flag

```
$ oc project --loglevel=10
$ oc get pods --loglevel=10
```

LAB 7

JBoss EAP Configuration

LAB 7 - VIEW LOGS

- View the Pod log for the application created in Lab 1

```
$ oc project lab1-<username>
$ oc get pods
NAME                  READY     STATUS      RESTARTS   AGE
kitchensink-1-7uwpl   1/1       Running     0          2d
kitchensink-1-build   0/1       Completed   0          2d

$ oc logs pod/kitchensink-1-7uwpl
```

- Change JBoss EAP log level for `org.infinispan` logger to `TRACE` via the container shell

```
$ oc rsh kitchensink-1-7uwpl

sh-4.2$ $JBOSS_HOME/bin/jboss-cli.sh --connect \
        --command="/subsystem=logging/logger=org.infinispan/:add(category=org.infinispan,level=TRACE,use-parent-handlers=true)"
sh-4.2$ $JBOSS_HOME/bin/jboss-cli.sh --connect --command="reload"
```

- Review the Pod log again (using `oc logs pod/<podname>`) and notice the log includes many `TRACE` entries for `org.infinispan`

JBOSS EAP CONFIGURATION

- JBoss EAP xPaaS configuration is stored in `$JBOSS_HOME/standalone/configuration/standalone-openshift.xml`
- When building with S2I, all files in the `configuration` subdirectory of the project source are copied to the JBoss EAP configuration dir. To provide your own config, it should be named `standalone-openshift.xml`
- All files in the `modules` source dir are copied to JBoss EAP modules dir
- Any `JAR`, `WAR`, and `EAR` in the dir specified by `ARTIFACT_DIR` (default `/target`) environment variable and also `deployment`'s source dir are copied to JBoss EAP deployments dir
- If environment variable `APP_DATADIR` defined on the build or BuildConfig, the content of the dir gets copied to JBoss EAP data dir

JBOSS EAP CONFIGURATION

`application/configuration`	copied →	`$JBOSS_HOME/standalone/configuration`
`application/modules`	copied →	`$JBOSS_HOME/modules`
`application/target`	copied →	`$JBOSS_HOME/standalone/deployments`
`application/deployments`	copied →	`$JBOSS_HOME/standalone/deployments`
`application/$APP_DATADIR`	copied →	`$JBOSS_HOME/standalone/data`

MAVEN CONFIGURATION

- JBoss EAP S2I builder by default enables Red Hat Maven repository for building the application. Applications can provide a custom `settings.xml` for Maven build through `application/configuration/settings.xml`
- The default Maven goal invoked by JBoss EAP S2I builder is

```
$ mvn -e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package
```

- Maven build can be configured through environment variables set on the build or `BuildConfig`

`MAVEN_MIRROR_URL`	URL of a Maven Mirror/repository manager to configure.
`MAVEN_ARGS`	*Overrides* `mvn` command used to build projects
`MAVEN_ARGS_APPEND`	*Appends* custom arguments to default `mvn` command

DATASOURCES

- Data Sources are automatically created based on the value of some environment variables set on the container or `DeploymentConfig`
- DB_SERVICE_PREFIX_MAPPING is the main environment variable which is a comma-separated list of triplets:

```
<name>-<database_type>=<PREFIX>
```

- `name` (lower-case) is used as the pool name
- `database_type` (lower-case) is used as driver name. PostgreSQL and MySQL supported
- `PREFIX` is the prefix in the environment variables used for retrieving username, password, database name, etc and configuring the datasource

DATASOURCES - EXAMPLE

- Datasource configuration example

```
DB_SERVICE_PREFIX_MAPPING=employee-postgresql=EMPLOYEEDB
```

- Creates a PostgreSQL datasource named `java:jboss/datasources/employee_postgresql`
- Configures the datasource based on environment variables it expects to be defined with the prefix `EMPLOYEEDB`

```
EMPLOYEEDB_JNDI
EMPLOYEEDB_USERNAME
EMPLOYEEDB_PASSWORD
EMPLOYEEDB_DATABASE
EMPLOYEEDB_TX_ISOLATION
EMPLOYEEDB_TX_MIN_POOL_SIZE
EMPLOYEEDB_TX_MAX_POOL_SIZE
```

- Expects the database service host and port be defined as environment variables

```
EMPLOYEE_POSTGRESQL_SERVICE_HOST
EMPLOYEE_POSTGRESQL_SERVICE_PORT
```

DATASOURCES - EXAMPLE

- When a service is created, OpenShift automatically sets the `<SERVICENAME>_SERVICE_HOST` and `<SERVICENAME>_SERVICE_PORT` environment variables on other containers within the project
- Creating a service named `<name>-<database_type>` would automatically set the required service environment variables for the datasource
- Multiple datasources can be defined at once

```
DB_SERVICE_PREFIX_MAPPING=employee-postgresql=EMPLOYEEDB,payroll-postgresql=PAYROLLDB
```

INCREMENTAL BUILDS

- Incremental builds allow reusing artifacts from previously-built images to reduce the build time
- Specially useful with Maven builds to skip downloading artifacts on every build

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: kitchensink
  ...
spec:
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jboss-eap64-openshift:latest
        namespace: openshift
      incremental: true
    type: Source
...
```

LAB 7 - JBOSS EAP APP DEPLOYMENT

- Create a new application based on the `eap70-mysql-s2i` template and the `kitchensink` Git repository. This template is more complex than the basic one.

```
$ oc new-project lab7-<username>
$ oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project -q)
$ oc create -f \
https://raw.githubusercontent.com/jboss-openshift/application-templates/master/secrets/eap7-app-secret.json

$ oc new-app eap70-mysql-s2i \
        --param=APPLICATION_NAME=todolist \
        --param=HTTPS_NAME=jboss \
        --param=HTTPS_KEYSTORE_TYPE=jks \
        --param=HTTPS_PASSWORD=mykeystorepass \
        --param=JGROUPS_ENCRYPT_NAME=secret-key \
        --param=JGROUPS_ENCRYPT_PASSWORD=password
```

- Several objects are created (image stream, build config, deployment config, services, routes) and the build is kicked off (you can follow with `oc logs -f bc/todolist` or watch the build from the OpenShift web console)

LAB 7 - JBOSS EAP APP DEPLOYMENT

- Explore the newly created project.
- A MySQL pod is deployed, along with a pod to run the Todo list EAP app.
- An HTTPS (secure) and HTTP service is created, each of which talk to the same application pod, using the self-signed certificates and passwords supplied.
- Routes are created for both secure and insecure variants.
- A database connection and JBoss EAP Datasource are established using the DB_SERVICE_PREFIX_MAPPING mechanism described earlier.

```
$ oc env dc/todolist --list | grep DB
...
DB_SERVICE_PREFIX_MAPPING=todolist-mysql=DB
DB_JNDI=java:jboss/datasources/TodoListDS
DB_USERNAME=userVGr
DB_PASSWORD=MONLi5nC
DB_DATABASE=root
...
```

LAB 7 - JBOSS EAP APP DEPLOYMENT

- Once the build completes, click on the secure route in the OpenShift web console, accept the self-signed certificate warning, and ensure the app is running.
- Create some sample TODO list entries by filling out the form and clicking *Submit*
- Log into the database pod and view the created entries in the database:

```
$ oc get pods|grep mysql
$ oc rsh todolist-mysql-1-12x37

sh-4.2$ echo 'select * from todo_entries' | \
        mysql -h $HOSTNAME -u $MYSQL_USER --password=$MYSQL_PASSWORD $MYSQL_DATABASE

Id                   summary            description
4812873360712653632  A sample entry      The description of the entry
-2901052661402745487  Another entry      Another description
```

APPLICATION CONFIGURATION

APPLICATION CONFIGURATION

- Applications decide what configuration mechanism they use
- DeploymentConfig can set environment variables on containers

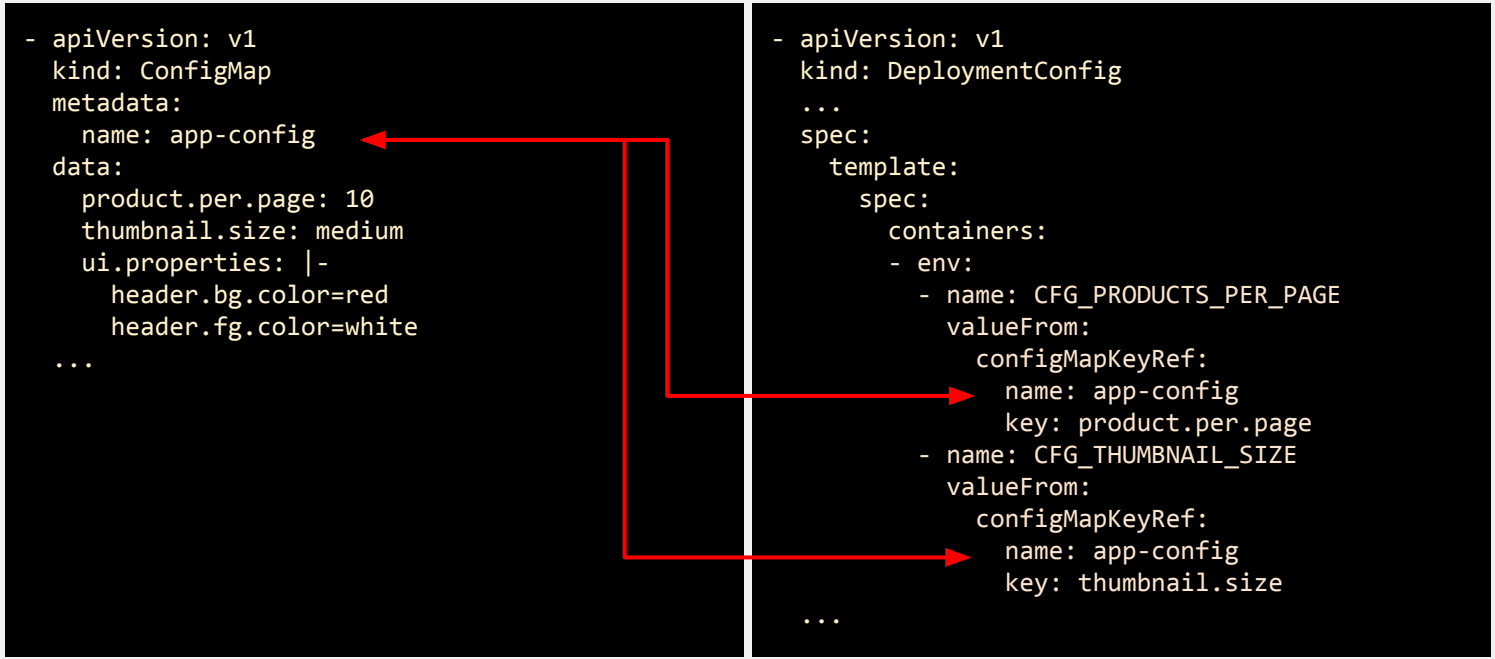
```
- apiVersion: v1
  kind: DeploymentConfig
  ...
  spec:
    template:
      spec:
        containers:
        - env:
          - name: CFG_PRODUCTS_PER_PAGE
            value: ${PRODUCT_PER_PAGE}
          - name: CFG_THUMBNAIL_SIZE
            value: ${THUMBNAIL_SIZE}
  ...
```

- Environment variables are useful but spread the configuration across many containers

APPLICATION CONFIGURATION

- Applications can retrieve configuration from external sources without OpenShift
- OpenShift 3.2 introduces ConfigMaps
  - Simplifies externalizing application configurations
  - Central configuration management for applications
  - Decouples configuration values from the container
- ConfigMaps can be consumed to
    - Populate environment variables
    - Set command line arguments in containers
    - Populate properties file in volumes

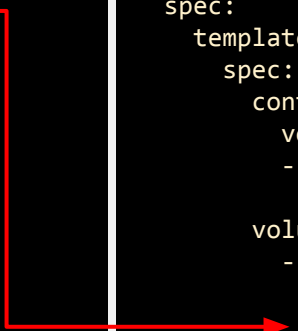
APPLICATION CONFIGURATION



APPLICATION CONFIGURATION

```
- apiVersion: v1
  kind: ConfigMap
  metadata:
    name: app-config
  data:
    product.per.page: 10
    thumbnail.size: medium
    ui.properties: |-
      header.bg.color=red
      header.fg.color=white
  ...
```

```
- apiVersion: v1
  kind: DeploymentConfig
  ...
  spec:
    template:
      spec:
        containers:
          volumeMounts:
          - name: config-volume
            mountPath: /etc/config
        volumes:
          - name: config-volume
            configMap:
            name: app-config
  ...
```



redhat.

You Made It!

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos