OPTUM™

Optum Global Solutions

https://oneconnect.uhg.com/docs/DOC-53617

Open Shift Enterprise V3

Day 02

Training Presentation for OpenShift Enterprise

Sreejith Kaimal

February 6, 2016

More Terminology ..

PROJECT

- A **project** allows a community of users to organize and manage their content in isolation from other communities.
- A project in Openshift contains multiple objects to make up a logical application.
- Can have separate name, display name and description.
- Each project scopes its own set of objects, policies , constraints & Service accounts.

TEMPLATE

- A template describes a set of objects that can be parameterized and processed
- It produces a list of objects for creation by OpenShift.
- You can define new templates to make it easy to recreate all the objects of your application.

ROUTE

- An route is a way to expose a service by giving it an externally-reachable hostname.
- Each route consists of a route name, service selector, and (optionally) security configuration.
- A defined route and the endpoints identified by its service can be consumed by a router.
- Router also provides load balancing features.

VOLUME

- Volumes are mounted file systems available to pods and their containers
- Volumes may be backed by a number of host-local or network attached storage endpoints.
- Persistent volume can be attached to a pod.
- Similar to a mount point.

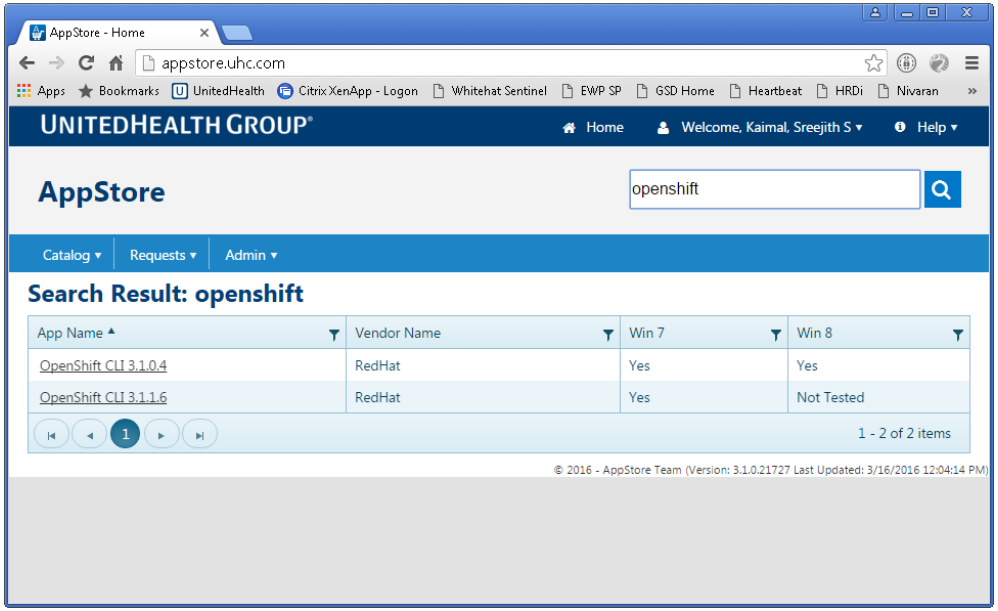
Request CLI

Request Installation of OpenShift CLI tool via appstore.

1. The OpenShift CLI exposes commands for managing your applications and tools to interact with each component within OSE.
2. Authorization is handled in the OpenShift policy engine.
3. The developer CLI uses the **oc** command, which is used for project-level operations.
4. The administrator CLI uses the **oadm** command for more advanced, administrator operations.

Various object types that CLI supports is listed below

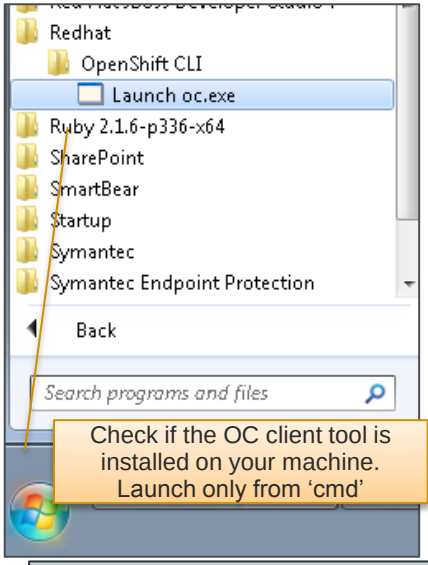
Object Type	Abbreviated Version
build	
buildConfig	bc
deploymentConfig	dc
event	ev
imageStream	is
imageStreamTag	istag
imageStreamImage	isimage
job	
LimitRange	limits
node	
pod	po
ResourceQuota	quota
replicationController	rc
secrets	
service	svc
ServiceAccount	serviceaccounts
persistentVolume	pv
persistentVolumeClaim	pvc



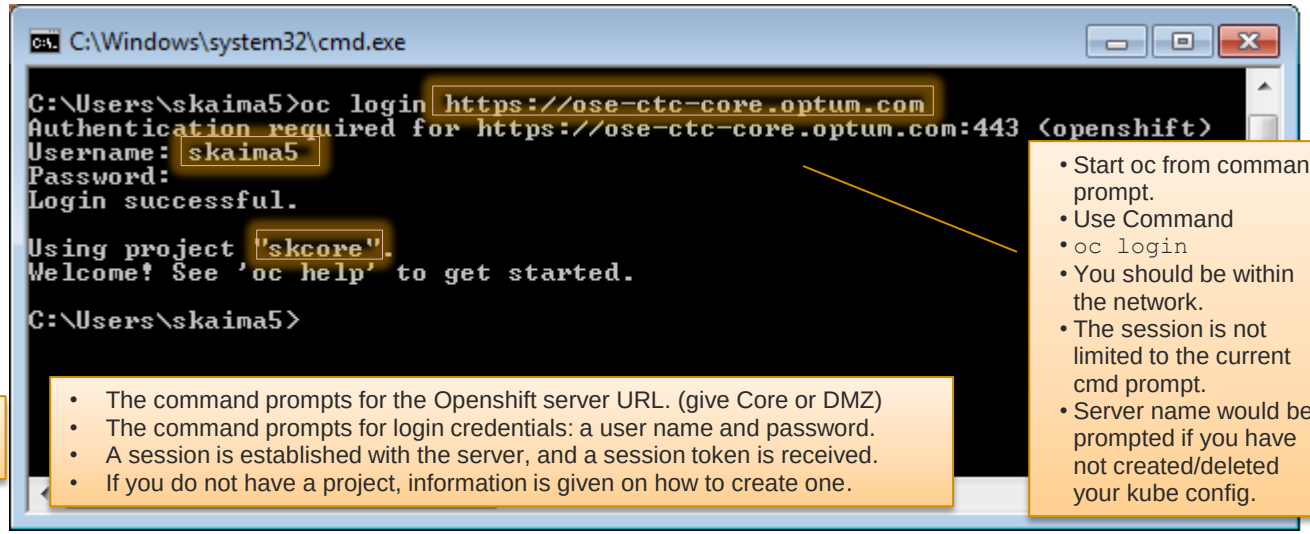
The CLI is ideal in situations where you are:

1. Working directly with project source code.
2. Scripting OpenShift operations.
   - Using templates , automating custom environment creation etc.
   - Update environment variables
   - Update labels on objects etc.
3. Restricted by bandwidth resources and cannot use the web console.
4. View details about containers that are in a stuck state.
5. Capture build logs etc

Setup CLI

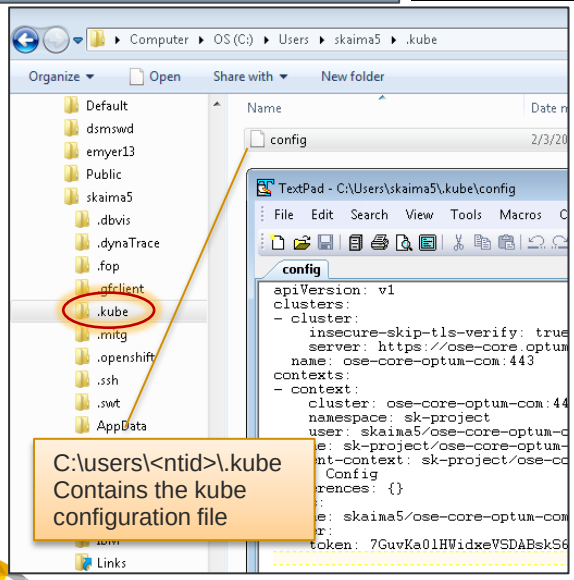


Check if the OC client tool is installed on your machine. Launch only from 'cmd'

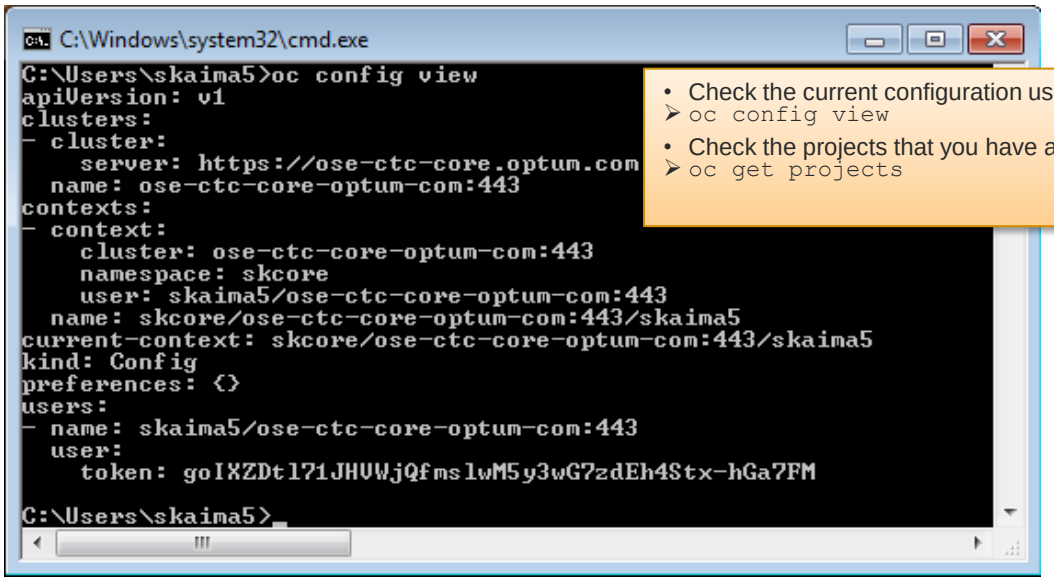


- The command prompts for the Openshift server URL. (give Core or DMZ)
- The command prompts for login credentials: a user name and password.
- A session is established with the server, and a session token is received.
- If you do not have a project, information is given on how to create one.

- Start oc from command prompt.
- Use Command
- `oc login`
- You should be within the network.
- The session is not limited to the current cmd prompt.
- Server name would be prompted if you have not created/deleted your kube config.



C:\users\<ntid>\.kube Contains the kube configuration file



- Check the current configuration using
  - `oc config view`
- Check the projects that you have access to.
  - `oc get projects`

https://docs.openshift.com/enterprise/3.1/cli_reference/get_started_cli.html

Manage CLI Profiles

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://ose-core.optum.com:443
  name: ose-core-optum-com:443
- cluster:
    insecure-skip-tls-verify: true
    server: https://ose-dmz.optum.com:443
  name: ose-dmz-optum-com:443
- cluster:
    insecure-skip-tls-verify: true
    server: https://ose-core.optum.com
  name: v3core
- cluster:
    insecure-skip-tls-verify: true
    server: https://ose-dmz.optum.com
  name: v3dmz
contexts:
- context:
    cluster: ose-dmz-optum-com:443
    namespace: sk-dmz
    user: skaima5/ose-dmz-optum-com:443
  name: sk-dmz/ose-dmz-optum-com:443/skaima5
- context:
    cluster: ose-core-optum-com:443
    namespace: sk-project
    user: skaima5/ose-core-optum-com:443
  name: sk-project/ose-core-optum-com:443/skaima5
- context:
    cluster: v3core
    user: skaima5/ose-core-optum-com:443
  name: skv3core
- context:
    cluster: v3dmz
    namespace: sk-dmz
    user: skaima5/ose-dmz-optum-com:443
  name: skv3dmz
current-context: sk-dmz/ose-dmz-optum-com:443/skaima5
kind: Config
preferences: {}
users:
- name: skaima5/ose-core-optum-com:443
  user: {}
- name: skaima5/ose-dmz-optum-com:443
  user: {}
```

The **clusters** section defines connection details for Openshift clusters, including the address for their master server.
A cluster is nothing but a server definition.

A **context** is a mapping of server to user & project.
You could have multiple contexts within the same server.
The namespace points to the project that you are using.

The **current context** parameter points to the current server/project/user that is being used.
All oc commands will use this configuration.
Ps: *if you are switching servers, you need to do a logout & explicit login*.

The **users** section holds the user credentials for the logged in or previously logged user.
When the user is logged in, it will contain a token key.

OpenShift allows multiple CLI configurations
Contexts allow you to easily switch between multiple users across multiple OpenShift servers, or clusters, when using issuing CLI operations. Nicknames make managing CLI configuration easier by providing short-hand references to contexts, user credentials, and cluster details.

Command	Usage
set-credentials	oc config set-credentials <username>
set-cluster	oc config set-cluster <cluster_nickname> --server=https://ose-core.optum.com --insecure-skip-tls-verify=true
set-context	oc config set-context <context_nickname> --cluster=<cluster_nickname> --user=<user_nickname> --namespace=<namespace>
use-context	oc config use-context <context_nickname>

Selecting a context to connect to OSE

There are two environments for OSE, so its better to configure both on your CLI config if you want to switch between them , here is how to do it.

1. Set Cluster Settings
   - `oc config set-cluster oseDmz --server=https://ose-ctc-dmz.optum.com --insecure-skip-tls-verify=true`
2. Set Context Settings
   - `oc config set-context oseDmzCtx --cluster=oseDmz --user= skaima5/ose-dmz-optum-com:443 --namespace=dmz-proj-name`
3. Switch context
   - `If you are switching context within the same cluster, then you can do in the same session.`
   - `If you are switching context across clusters, then you need to logout and then switch context and then login again.`
4. Switch context , the context switch takes care of pointing to the right cluster.
   - `oc config use-context oseDmzCtx`
5. You can repeat steps 1 & 2 for as many servers you want to add. (e.g, ose-dmz and ose-core)

Generic OC command instruction format

- `oc <action> <object_type> <object_name_or_id>`

Create a project

- `oc new-project hello-world --description="This is an example project to demonstrate OpenShift v3" --display-name="Hello World"`

Switch Project

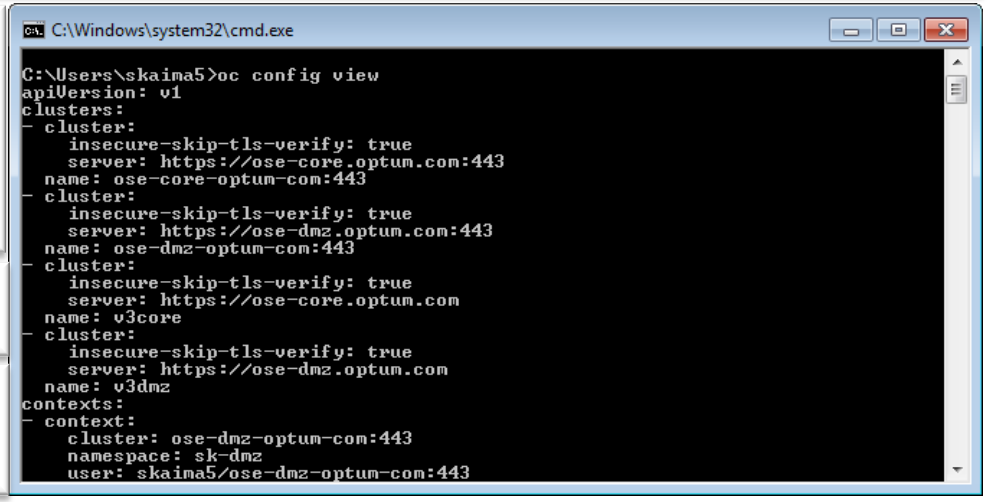
- `oc project hello-world`

Create an Application

- `oc new-app mysql`

Check some OpenShift object types via oc, this lists a generic help on object types.

- `oc types`



https://docs.openshift.com/enterprise/3.1/dev_guide/new_app.html

OC CLI Commands reference

Operation	Syntax
Get	`oc get <objecttype> <objectname>` `oc get pods`
Describe	`oc describe <objecttype> <objectid>` `Oc describe pod  jbosstom-1-17v5h`
Edit	`oc edit <objecttype>/<objectname>` `oc edit pod/jbosstom-1-17v5h`
Env	`oc env <objecttype>/<objectname> <envvar>=<value>` `Some objects do not allow adding env variables directly.`
Label	`oc label <objecttype> <objectname> <label>` `oc label pods foo unhealthy=true`

Stop	`oc stop -f <file_path>`
	`oc stop <object_type> <object_name>`
	`oc stop <object_type> -l <label>`
	`oc stop all -l <label>`

Delete	`oc delete -f <file_path>`
	`oc delete <object_type> <object_name_or_id>`
	`oc delete <object_type> -l <label>`
	`oc delete all -l <label>`

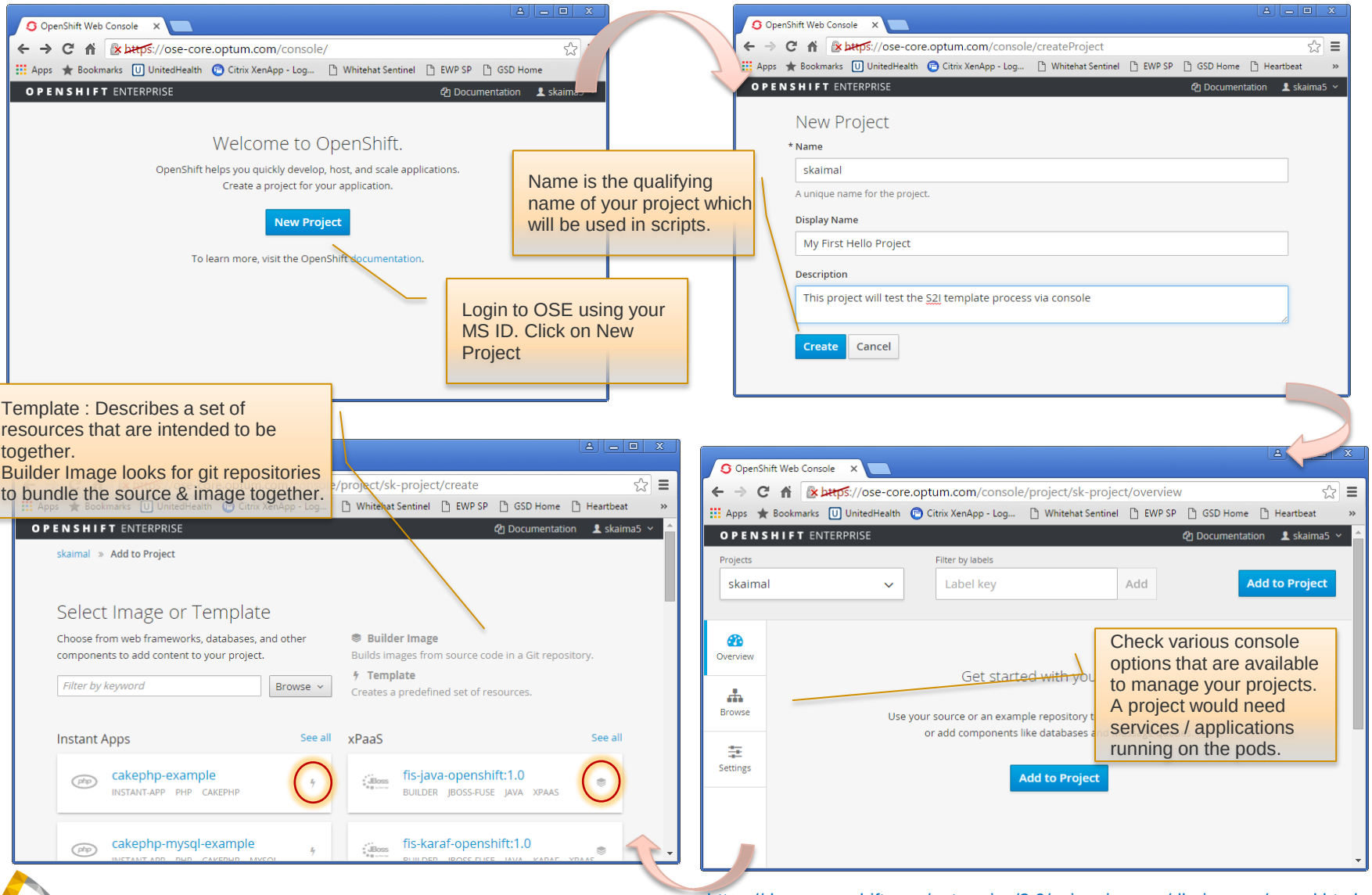
Expose	`oc expose <object_type> <objectname>`
Project	`oc project <project_name>`
New Project	`oc new-project <project_name>`

Operation	Syntax
Logs	`oc logs -f <podname> <containername>`
Exec	`oc exec <podname> [-c <containerID>] <command>` `oc exec -it --pod=jbosstom-1-17v5h /bin/bash`
RSH	`oc rsh <podname>` `oc rsh jbosstom-1-17v5h`
port-forward	`oc port-forward <pod_ID> <first_port_ID> <second_port_ID>`
Run Proxy	`oc proxy --port=<port_ID> --www=<static_dir>`
Policy	`Manage Policy Authorization.` `oc policy [--options]`
Scale	`oc scale <object_type> <object_id> --replicas=<Num_replicas>`
Start Build	`oc start-build --from-build=<build_name>` `oc start-build <buildConfig_name>`
View Build Log	`oc build-logs <build_name>`
Deploy	`oc deploy <deploymentconfig>`

Login	`oc login [-u=<username>]`
	`[-p=<password>]`
	`[-s=<server>]`
	`[-n=<project>]`
	`[--certificate-authority=</path/to/file.crt>|--insecure-skip-tls-verify]`

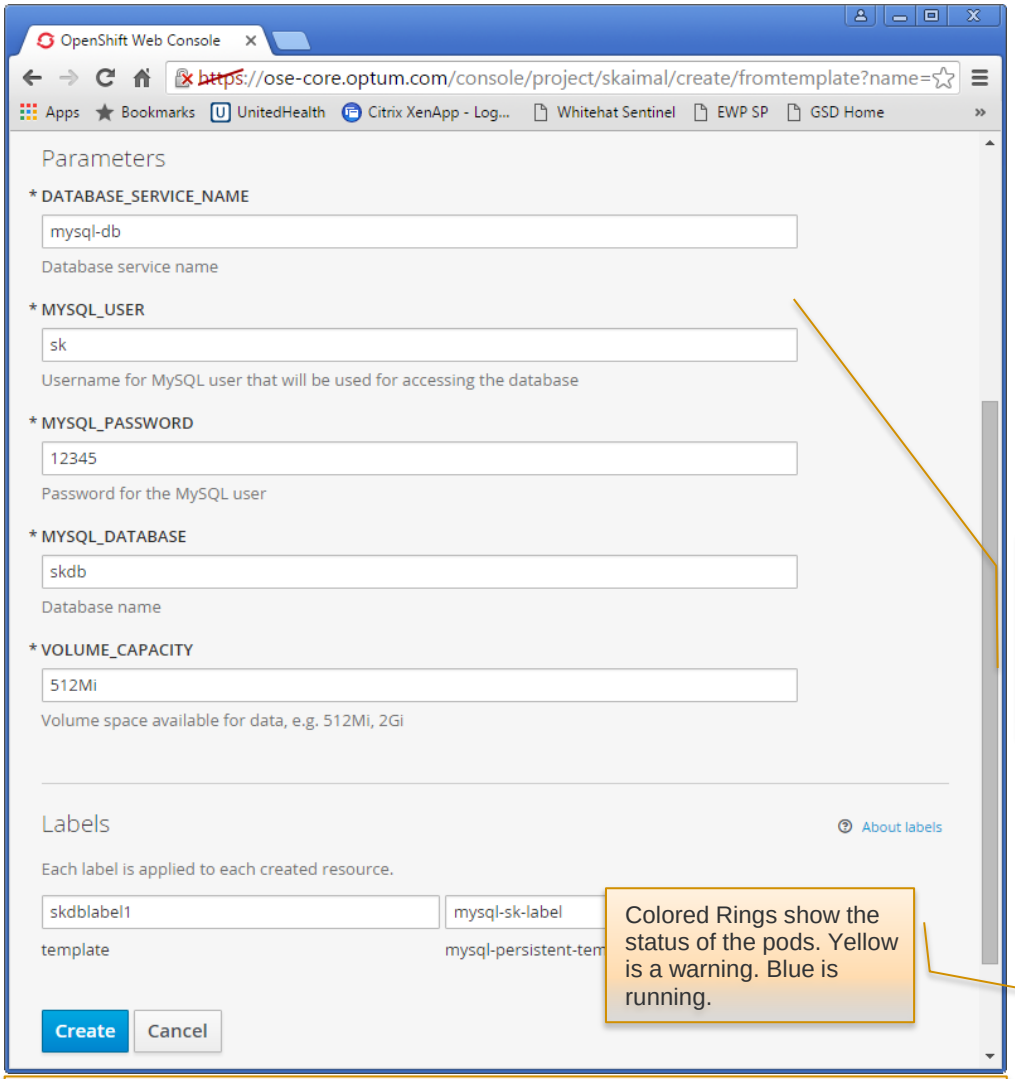
https://docs.openshift.com/enterprise/3.0/cli_reference/basic_cli_operations.html

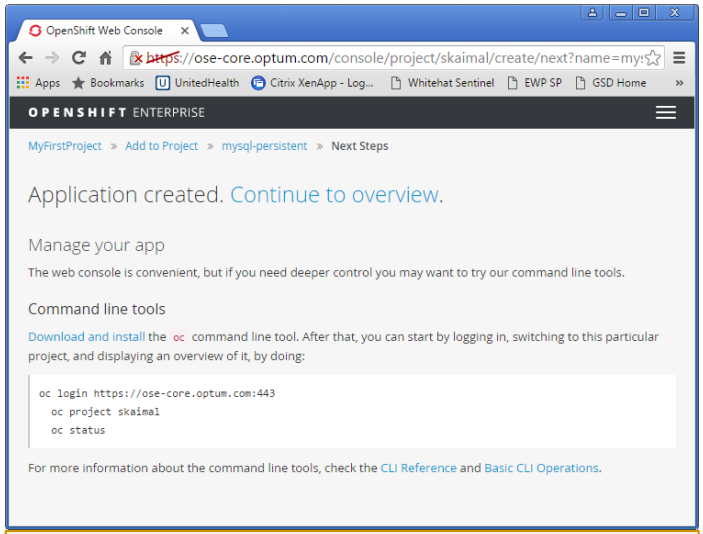
OSE Console



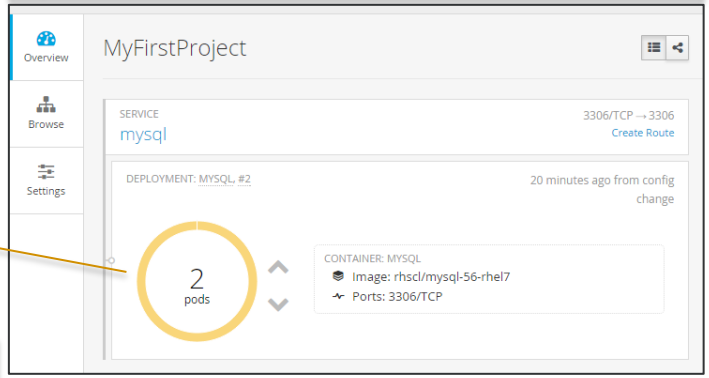
https://docs.openshift.com/enterprise/3.0/using_images/db_images/mysql.html

OSE Console





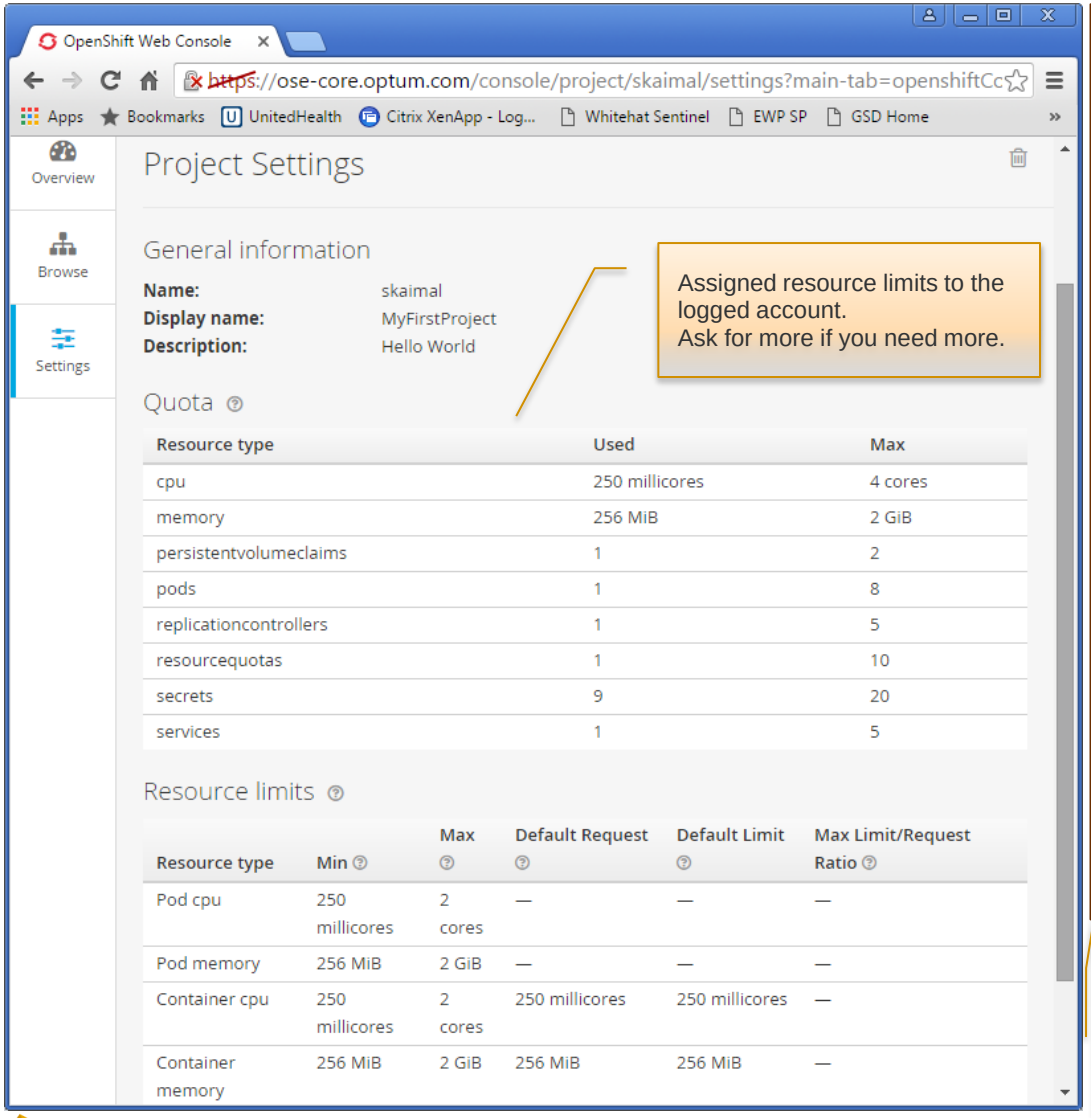
While creating an application from a Template or an Image, the form fields are different.
An Image build expects for a git repo which it can use to bundle with the image.
A template build simply creates from a standard template file– Good for creating a DB or MQ or anything similar project specific.



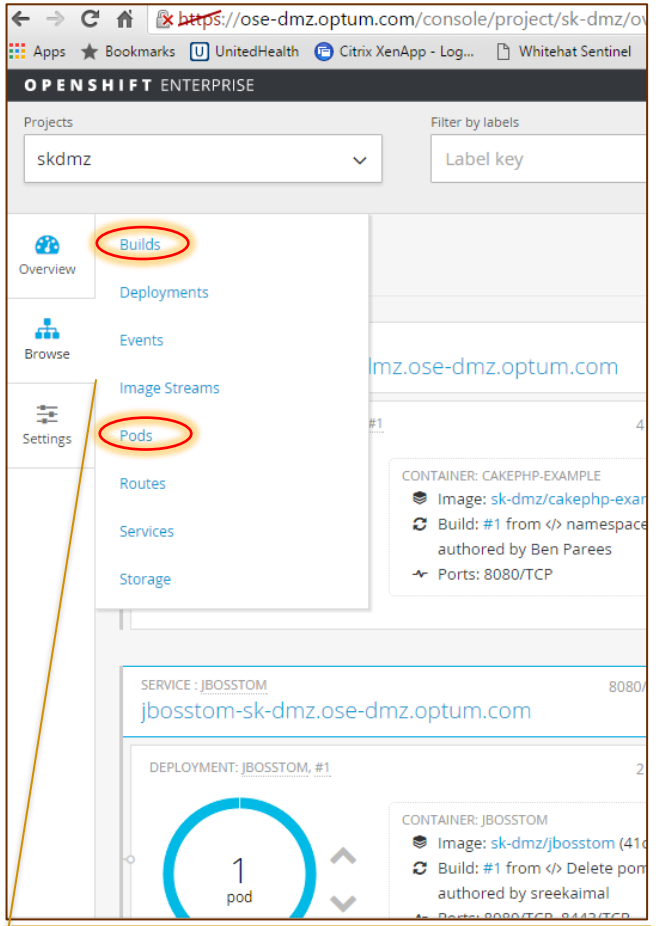
Colored Rings show the status of the pods. Yellow is a warning. Blue is running.

Many features are not ready yet (as on Feb 2016), so errors could be environmental or specific to application. Please use the oneconnect forum, if in doubt.

OSE Console



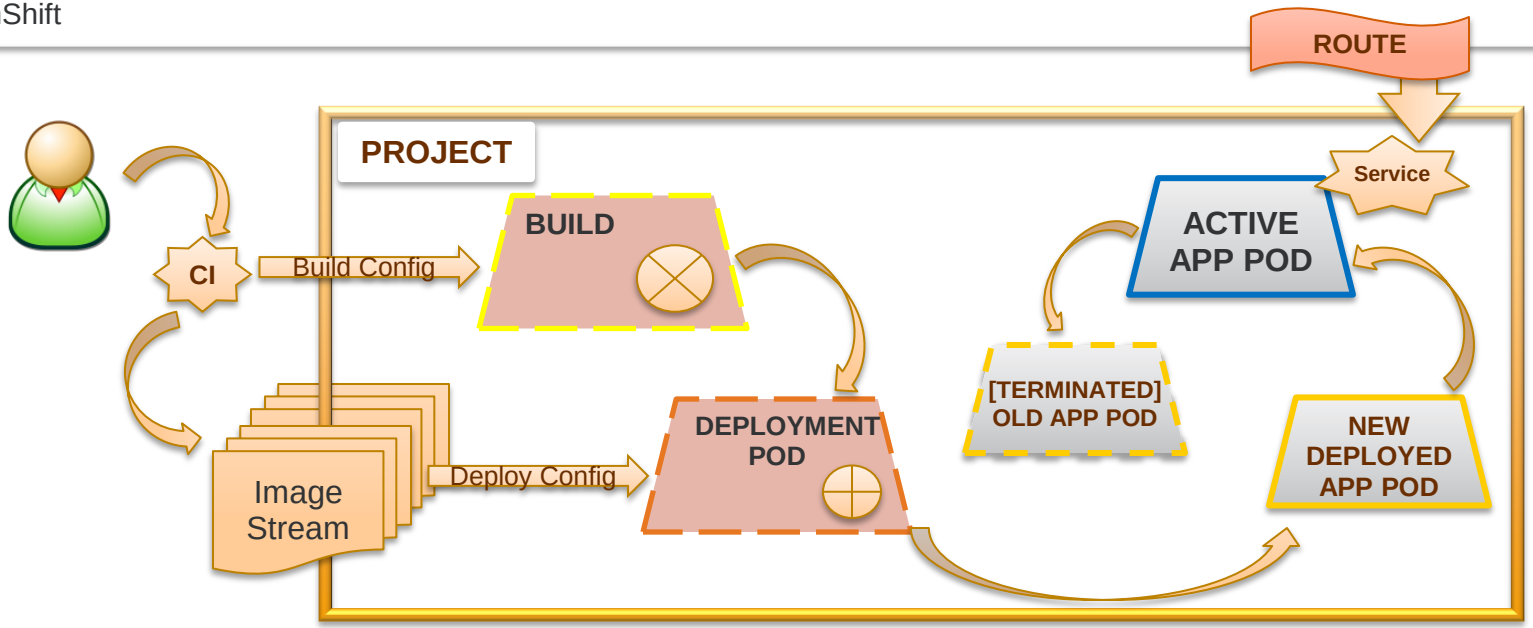
Assigned resource limits to the logged account.
Ask for more if you need more.



Each ‘Object’ that you add to the ‘Project’ creates its own configuration files. These configuration items trigger the changes.
You can see BuildConfig, Deployment, Image Stream, Pods, Routes , Services in the ‘Browse’ menu.
Deleting a config, triggers deletion of the object.

Core Objects

1. **Containers & Images** : Linux container technologies are lightweight mechanisms for isolating running processes so that they are limited to interacting with only their designated resources
2. **Pods**: The smallest compute unit that can be defined, deployed, and managed.
3. **Services** :Service serves as an internal load balancer. It identifies a set of replicated pods in order to proxy the connections it receives to them. Backing pods can be added to or removed from a service arbitrarily while the service remains consistently available.
4. **Projects** :Kubernetes namespace provides a mechanism to scope resources in a cluster.
5. **Users** :Regular users, System Administrators, Service accounts
6. **Builds**: the process of transforming input parameters into a resulting object (Source to Image)
7. **Image streams**: can be used to automatically perform an action, such as updating a deployment, when a new image is made available.
8. **Deployment**: creates a new replication controller and lets it start up pods. Replication controller ensures that a specified number of replicas of a pod are running at all time
9. **Routes** : is a way to expose a service by giving it an externally-reachable hostname
10. **Templates** describes a set of objects that can be parameterized and processed to produce a list of objects for creation by OpenShift



Build & Deploy

Process of 'not only' building 'code' + 'Applications' but also the environments.
It does the process of Transforming input parameters into a resulting object
A '*BuildConfig*' object is the definition of the entire build process.
OpenShift leverages Kubernetes by creating Docker containers from build images and pushing them to a Docker registry.
The OpenShift build system provides extensible support for build strategies that are based on selectable types specified in the build API. There are three build strategies available:

1. **Docker build**
   - Builds Docker images from a Dockerfile and a 'context'. A build's context is the files located in the specified PATH or URL.
   - The URL parameter can specify the location of a Git repository; the repository acts as the build context.
2. **Source-to-Image (S2I) build**
   - Tool for building reproducible Docker images.
   - It produces ready-to-run images by injecting application source into a Docker image and assembling a new Docker image.
   - New image incorporates the base image (the builder) and built source and is ready to use with the `docker run` command.
3. **Custom build**
   - The Custom build strategy allows developers to define a specific builder image responsible for the entire build process.
   - https://hub.docker.com/r/openshift/origin-custom-docker-builder/

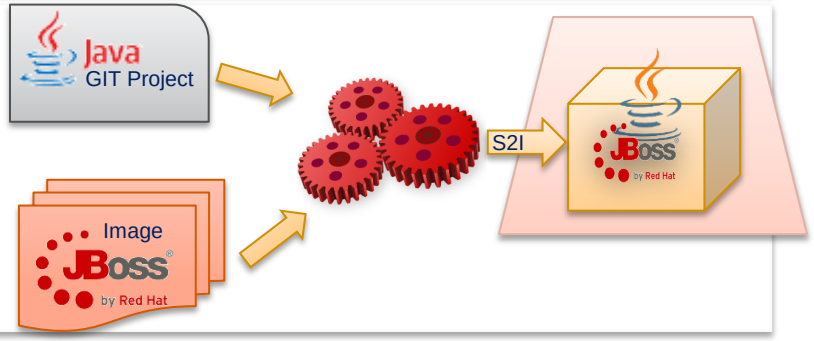
Source To Image Build (**S2I**)
'Framework' that makes it easy to write images that take application source code as an input and produce a new image that runs the assembled application as output.
The build process consists of the following three fundamental elements, which are combined into a final Docker image:

- Sources
- S2I scripts
- Builder image

The builder places 'Sources' & 'Scripts' inside the 'Image'

Build Process in S2I

1. Download the S2I Build Script.
2. Download the application source code that needs to be built (git)
3. Bundle source code as 'tar' file to be placed inside the Image(pom)
4. *Assemble -> Run Build -> Install* tar content into the Image sources.
5. Run script takes care of starting the application within the image/container.
6. Any errors in each step causes the build to fail.
7. *Usage* provides documentation on how to use the build.

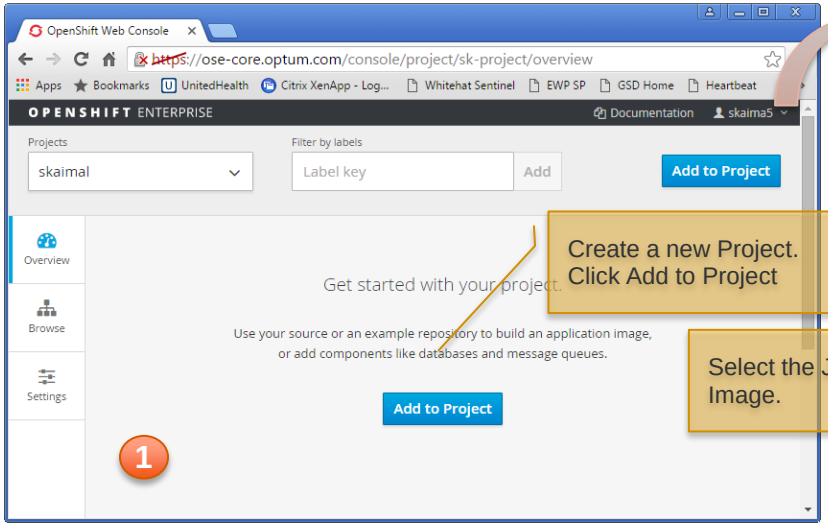


Build & Deploy

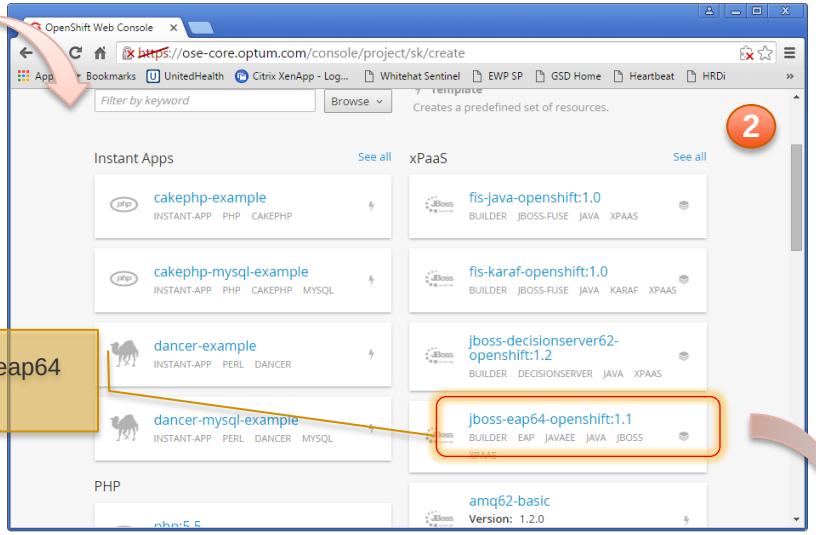
You can create a new OpenShift application using the web console **or** by running the `oc new-app` command from the CLI.
OpenShift creates a new application by specifying source code, images, or templates.
The new-app command looks for images on the local Docker installation (if available), in a Docker registry, or an OpenShift image stream.
If creating a **Source** build, **new-app** attempts to determine which language builder to use based on the presence of certain files in the root of the repository, e.g for a JEE project, it should have a *pom.xml* in the root folder.
The new-app creation triggers the following objects to be created.

Artifact	Description
BuildConfig	A BuildConfig is created for each source repository specified in the command line. The BuildConfig specifies the strategy to use, the source location, and the build output location.
ImageStream	An image stream can be used to automatically perform an action, such as updating a deployment, when a new image is made available. For BuildConfig, two Image Streams are usually created: one to represent the input image (the builder image in the case of Source builds or FROM image in case of Docker builds), and another one to represent the output image. If a Docker image was specified as input to new-app, then an image stream is created for that image as well.
DeploymentConfig	A DeploymentConfig is created either to deploy the output of a build, or a specified image.
Service	The new-app command attempts to detect exposed ports in input images. It uses the lowest numeric exposed port to generate a service that exposes that port. In order to expose a different port, after new-app has completed, simply use the oc expose command to generate additional services.

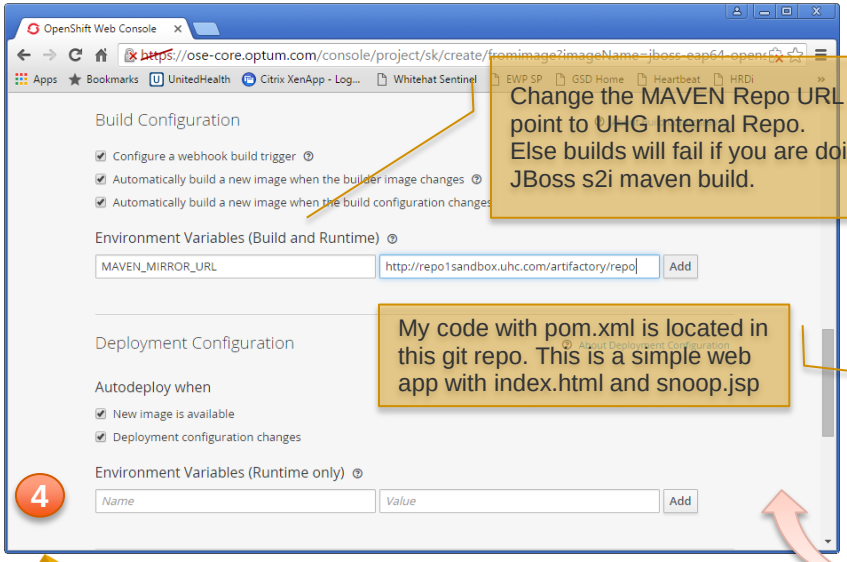
Build & Deploy your Application using S2I



Create a new Project.
Click Add to Project

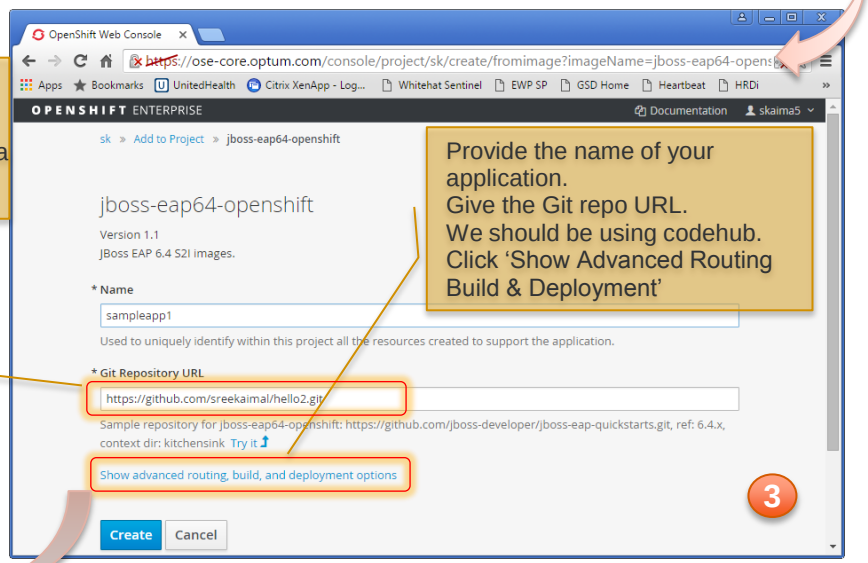


Select the JBoss eap64 Image.



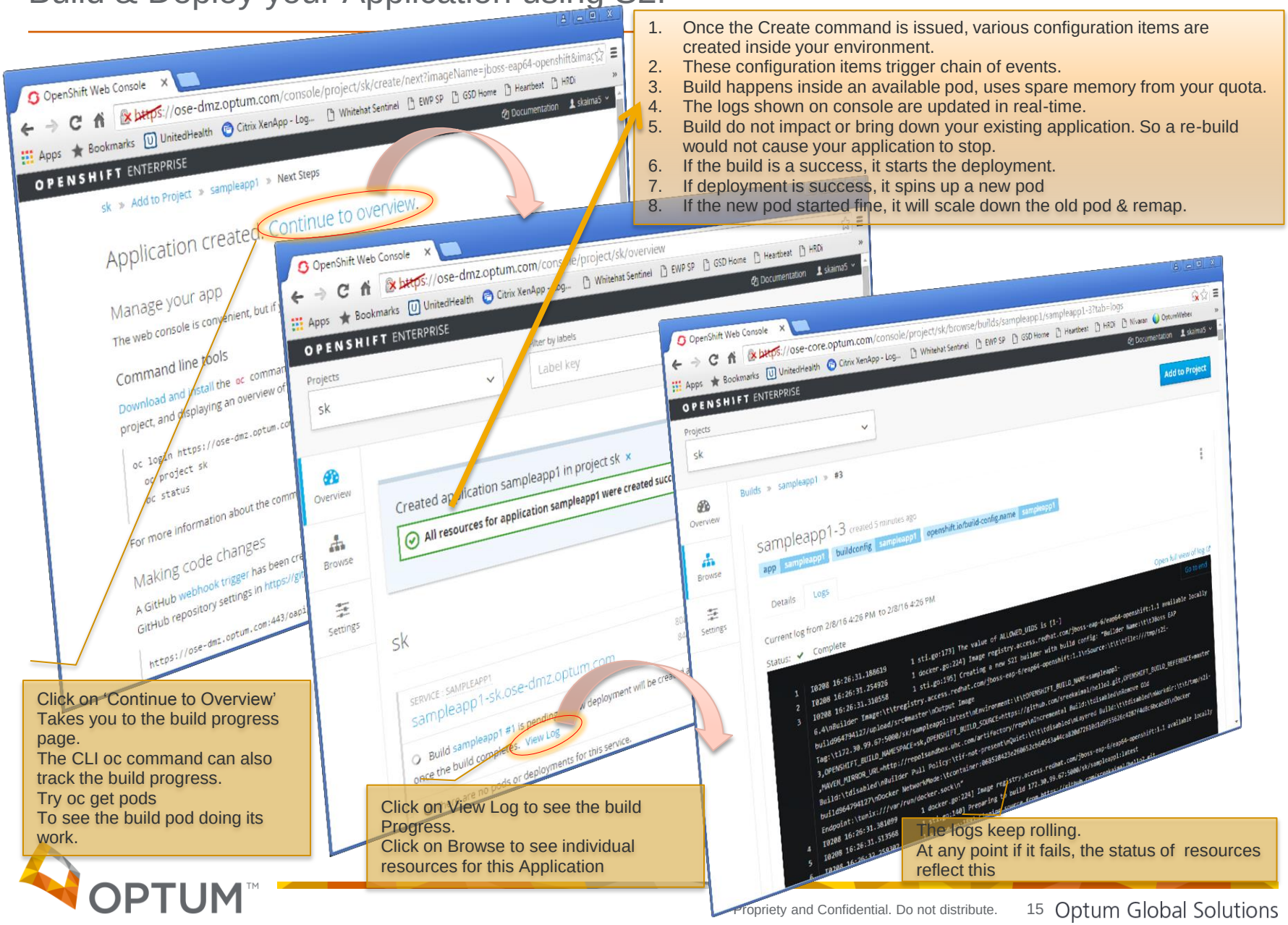
Change the MAVEN Repo URL to point to UHG Internal Repo.
Else builds will fail if you are doing a JBoss s2i maven build.

My code with pom.xml is located in this git repo. This is a simple web app with index.html and snoop.jsp



Provide the name of your application.
Give the Git repo URL.
We should be using codehub.
Click 'Show Advanced Routing Build & Deployment'

Build & Deploy your Application using S2I



1. Once the Create command is issued, various configuration items are created inside your environment.
2. These configuration items trigger chain of events.
3. Build happens inside an available pod, uses spare memory from your quota.
4. The logs shown on console are updated in real-time.
5. Build do not impact or bring down your existing application. So a re-build would not cause your application to stop.
6. If the build is a success, it starts the deployment.
7. If deployment is success, it spins up a new pod
8. If the new pod started fine, it will scale down the old pod & remap.

Click on 'Continue to Overview' Takes you to the build progress page.
The CLI oc command can also track the build progress.
Try oc get pods
To see the build pod doing its work.

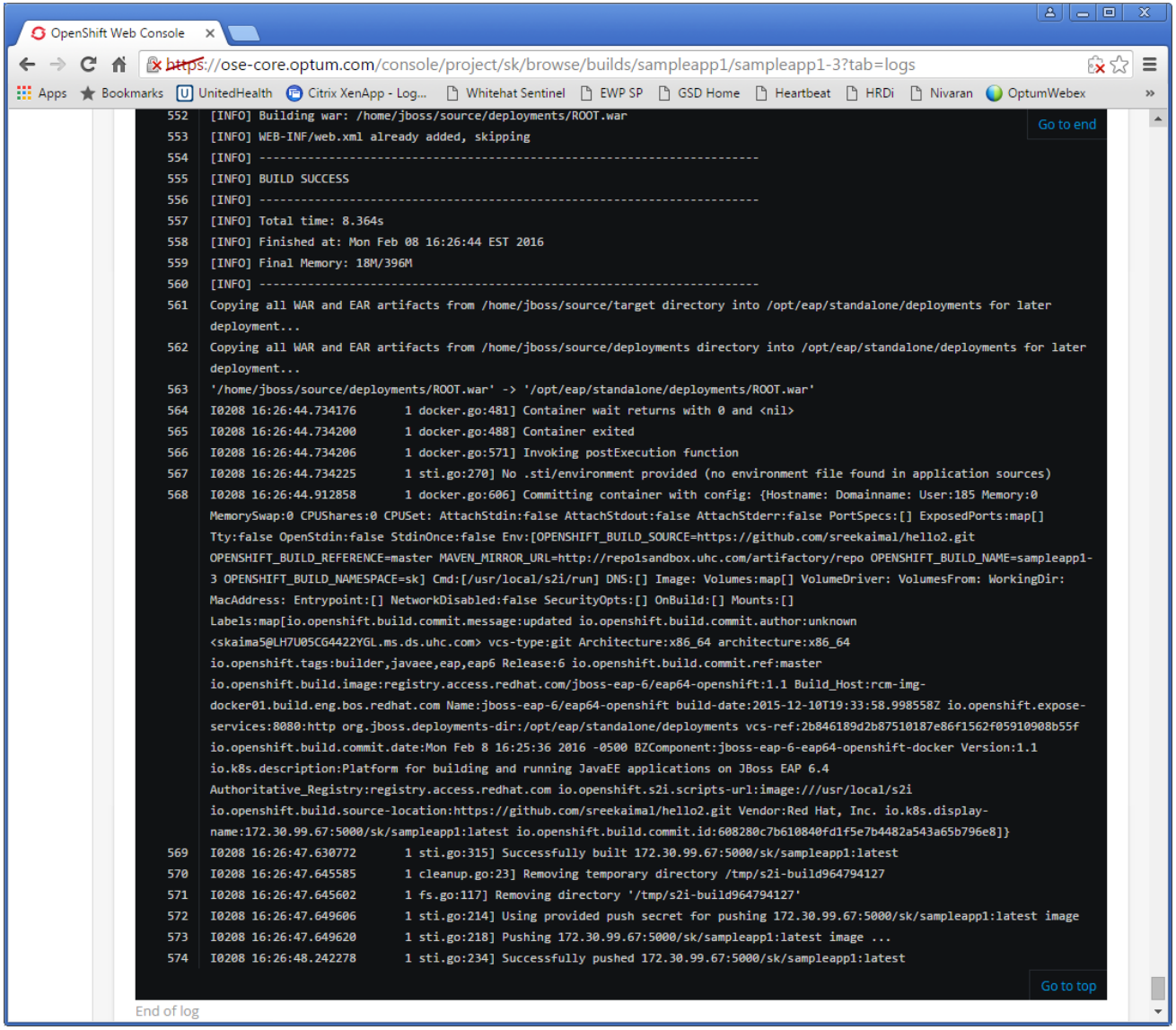
Click on View Log to see the build Progress.
Click on Browse to see individual resources for this Application

The logs keep rolling.
At any point if it fails, the status of resources reflect this

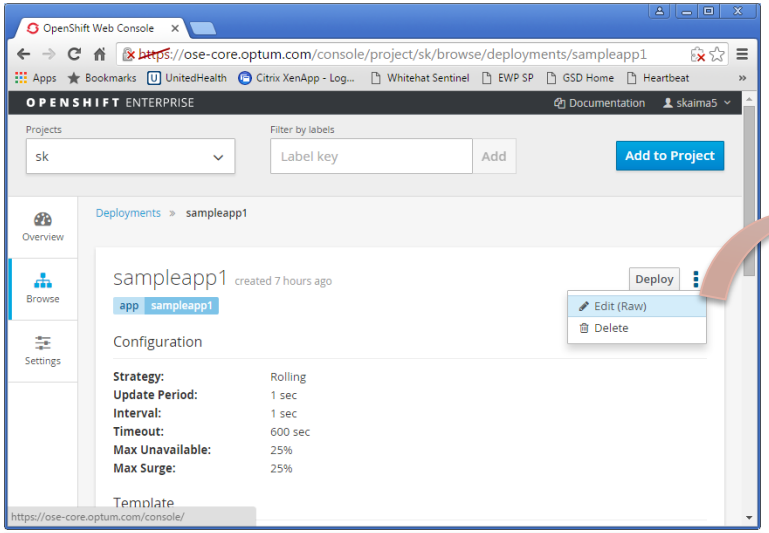
Build & Deploy your Application using S2I

- A successful build does not ensure a successful deployment.
- A deployment would request additional resources from your environment.
- A deployment starts another similar pod before it can bring down the existing pod.
- Lack of resources can cause the sequence of events to halt.
- All events are recorded in the events section. Some events would be very generic, like "*Back off restarting failed docker container*" and some might be specific like "*unable to admit pod without exceeding quota for resource memory*"
- If the container restarts often, it means that the image is not able to sustain within the pod.
- This could be an issue with the code or the server or lack of resources to run.
- By default each container gets 256MB of memory, which might not be enough for e.g,JBoss server. In this case, modifying the deployment configuration would be required.
- Check the settings section to know the current state of your OpenShift account.
- Check pod state & logs for more info..

```
C:/> oc logs <podname>
```



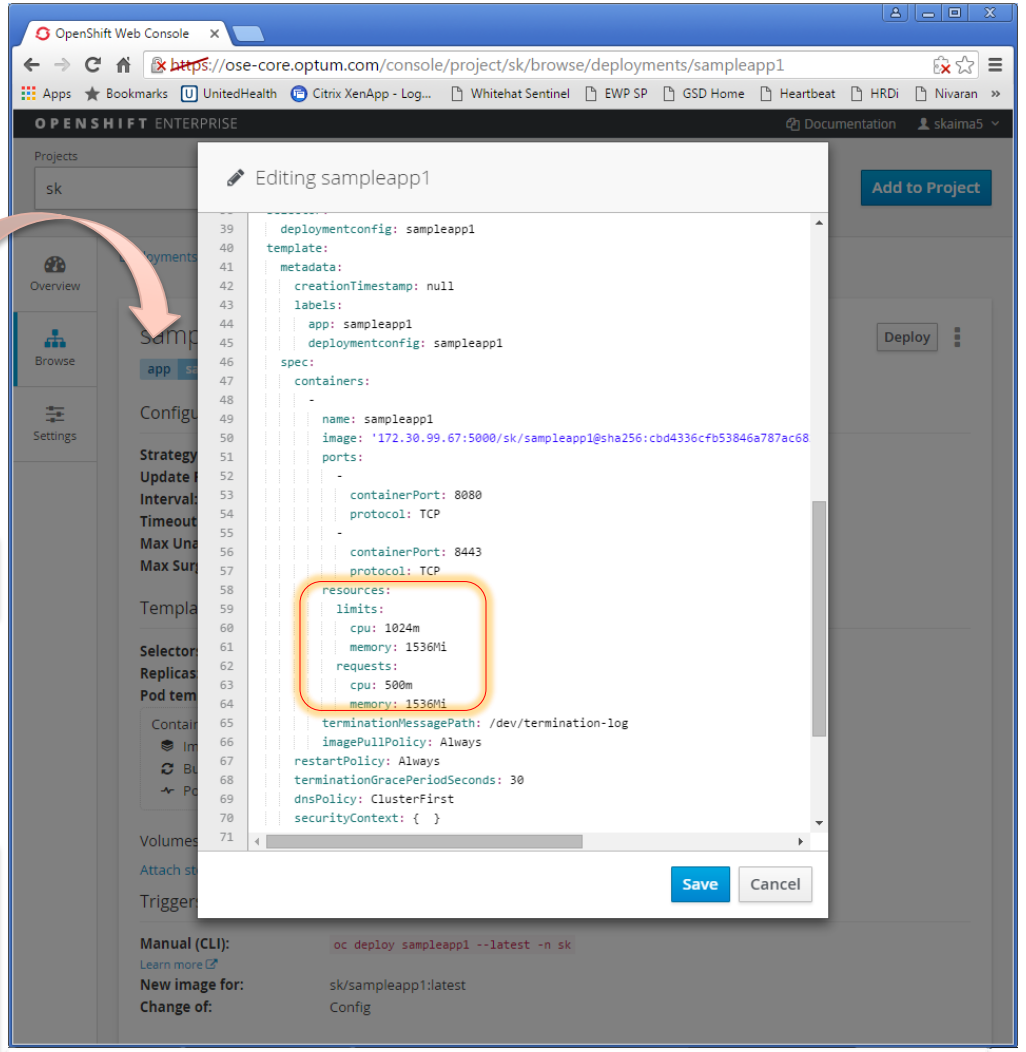
Edit Deployment Configuration



If the pod logs **do not** look something like below, it means server **did not** started successfully -

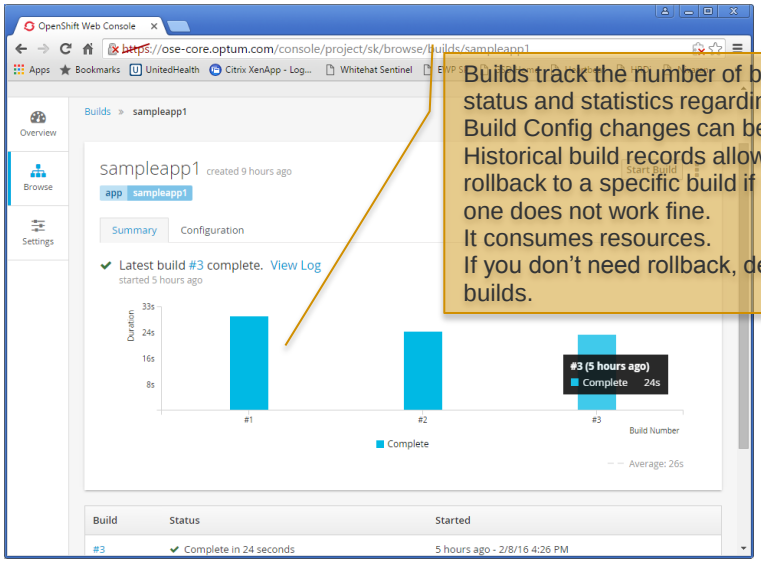
```
[org.jboss.as] (Controller Boot Thread) JBAS015961:
Http management interface listening on
http://0.0.0.0:9990/management
[org.jboss.as] (Controller Boot Thread) JBAS015951:
Admin console listening on http://0.0.0.0:9990
[org.jboss.as] (Controller Boot Thread) JBAS015874:
JBoss EAP 6.4.4.GA (AS 7.5.4.Final-redhat-4) started in
12186ms - Started
```

Edit the deployment config to increase the container resource allowance as shown in the screenshot.
Save and click the deploy button again.
Multiple deploys exhaust the resources e.g,replication controller.
Click on Browse->Deployments and next to your deployment, you will see a #X where x is number of deploy attempts. Click on that and delete old attempts.
Don't delete the deployment configuration itself.

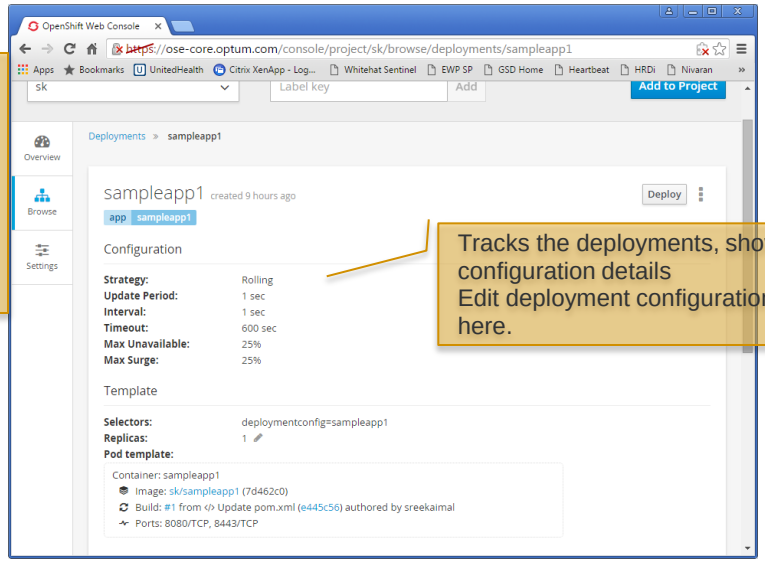


If the pod logs **do not** look something like below, it means server **did not** started successfully -

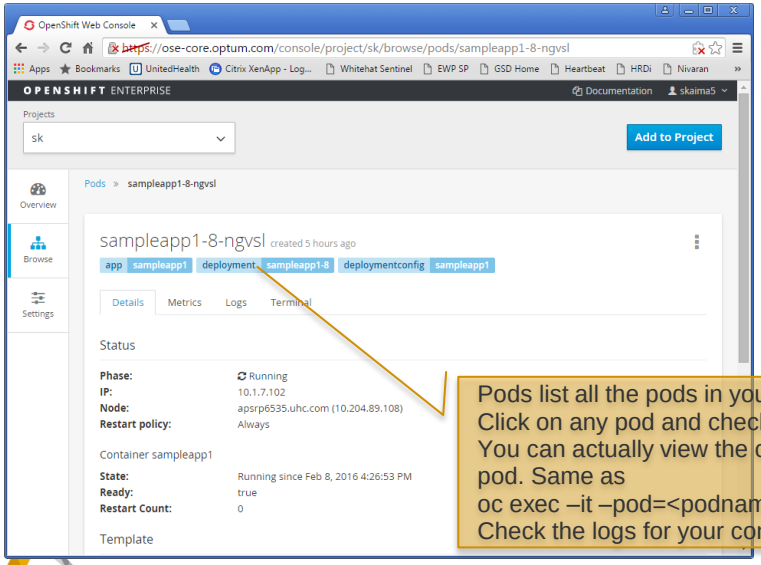
Checking Build , Routes, Deployment & Pods in project



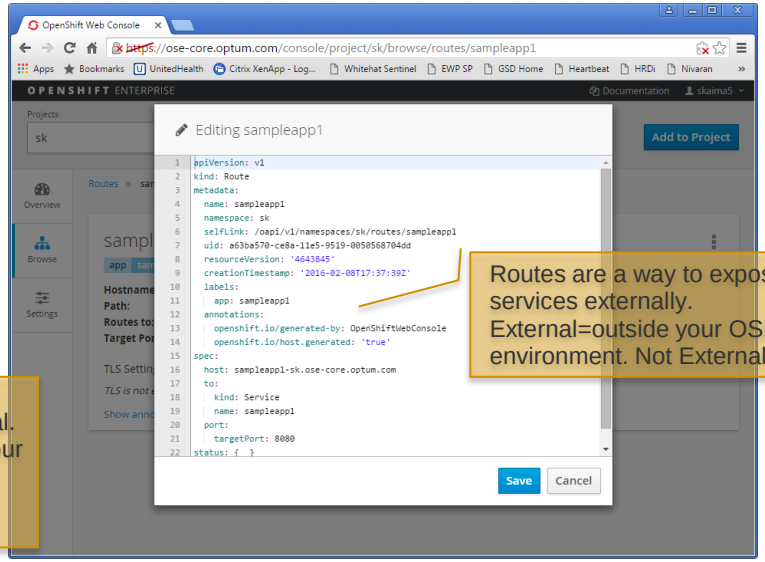
Builds track the number of builds, the status and statistics regarding the build.
Build Config changes can be made here.
Historical build records allow us to rollback to a specific build if the current one does not work fine.
It consumes resources.
If you don't need rollback, delete old builds.



Tracks the deployments, shows configuration details
Edit deployment configuration here.



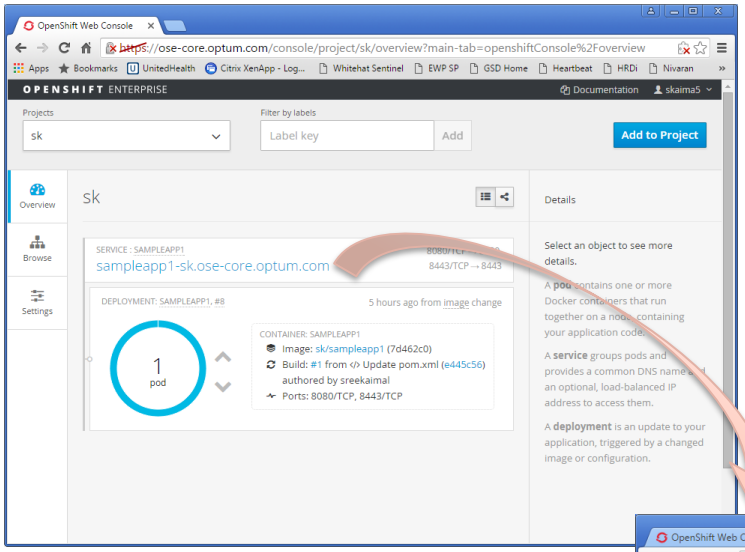
Pods list all the pods in your account.
Click on any pod and check the Terminal.
You can actually view the contents of your pod. Same as
oc exec –it –pod=<podname>
Check the logs for your container here.



Routes are a way to expose your services externally.
External=outside your OSE environment. Not External

Testing..

http://sampleapp1-sk.ose-core.optum.com/



1. Check if the pod status is Blue - consistently.
2. Check events to see if there are no container restarts being logged.
3. Check settings to see if there are no warning icons or max limits.
4. Check if logs are being generated in pod.

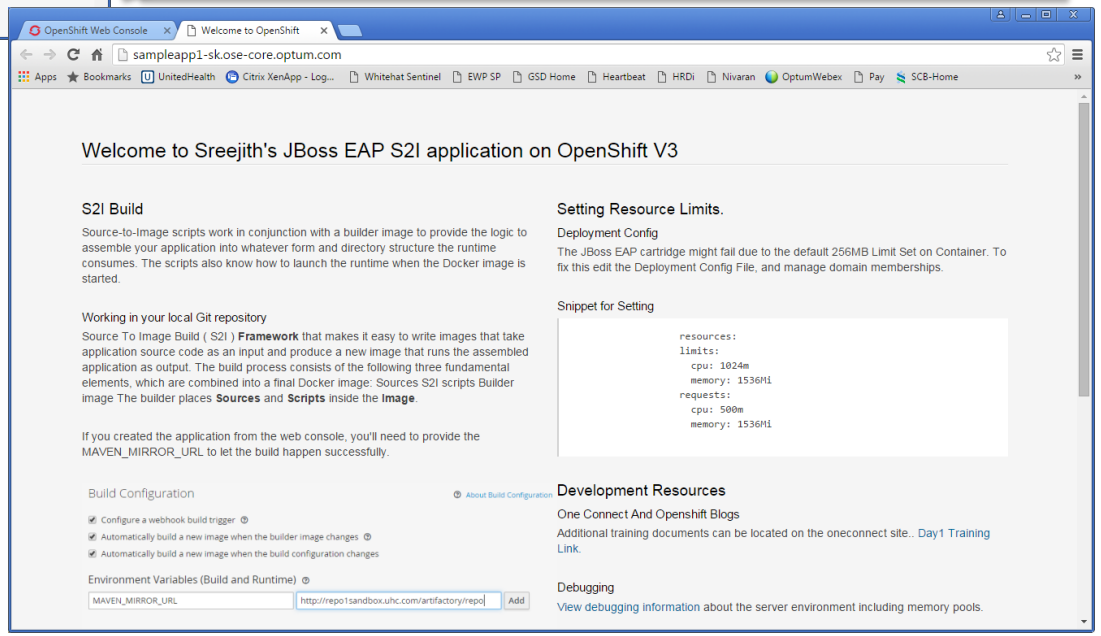
1. Try modifying the source git repo & hit build again.
2. In 'Overview' See how the build & deploy spins a new pod.
3. Check resource usages during a build.
4. If you are out of resources, then delete a few unused configuration items / failed pods.
5. Resource quota can be increased if required.
6. Always keep the container memory at a moderate level. We don't need to spin up a container with 4GB when we can get optimum performance with 2GB.
7. Remember this is horizontally scalable.

Next Steps

1. Create a Web Project in JBDS or eclipse.
2. Deploy locally on Jboss EAP 6.4 , test
3. Upload it to codehub
4. Build a S2I package
5. Deploy to OpenShift V3
6. Check what shows in overview when you build & deploy.

Help & Support available via Oneconnect
https://oneconnect.uhg.com/groups/openshift-enterprise-users-group

Support team can be reached at
OpenShiftEnterpriseSupport_DL@ds.uhc.com



OPTUM™

Optum Global Solutions

Thank you.

Sreejith Kaimal
Sr.Architect
Provider Portal Applications