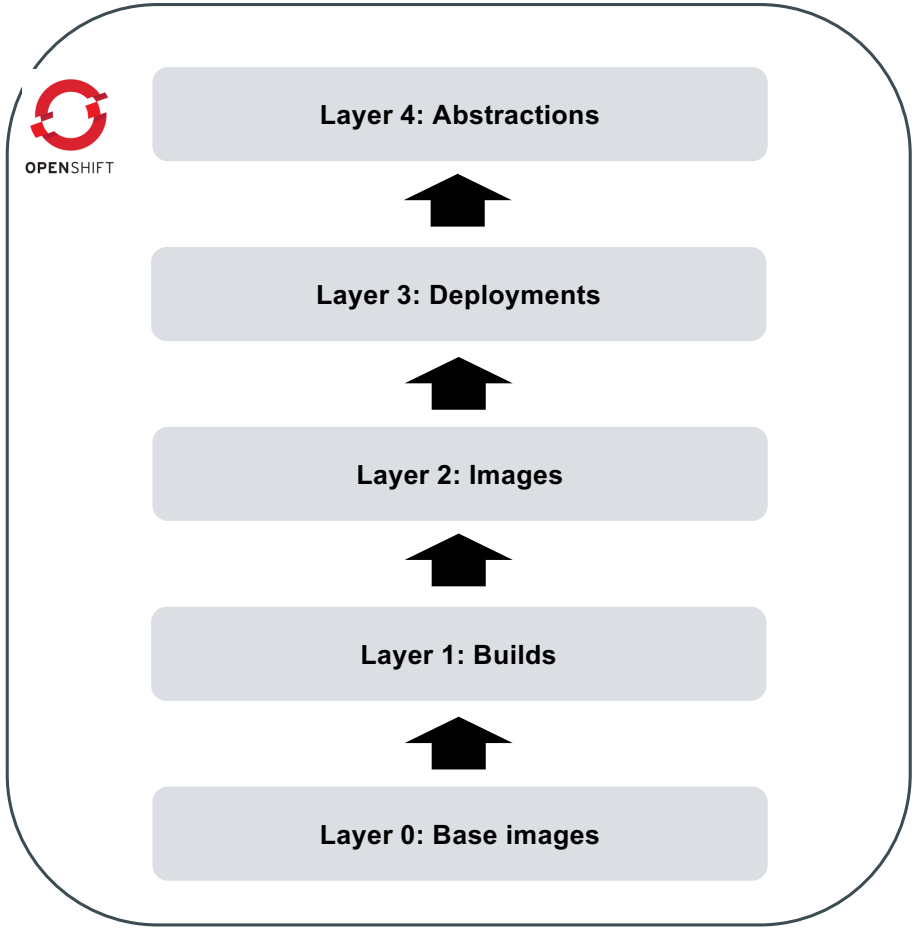
Deploying to OpenShift S2I and templates

WW Developer Advocate team

OpenShift Deployment – Developer view



Services, routes, PVCs, secrets etc

Defines what will run in OpenShift

Images created by build layer

Defines the builds required for the app

The base images used by the apps

Layer 0
Base image examples

- Base operating systems, specific runtimes, databases, application servers and more
  - Available as **Image Streams**, an enhanced set of metadata about each image

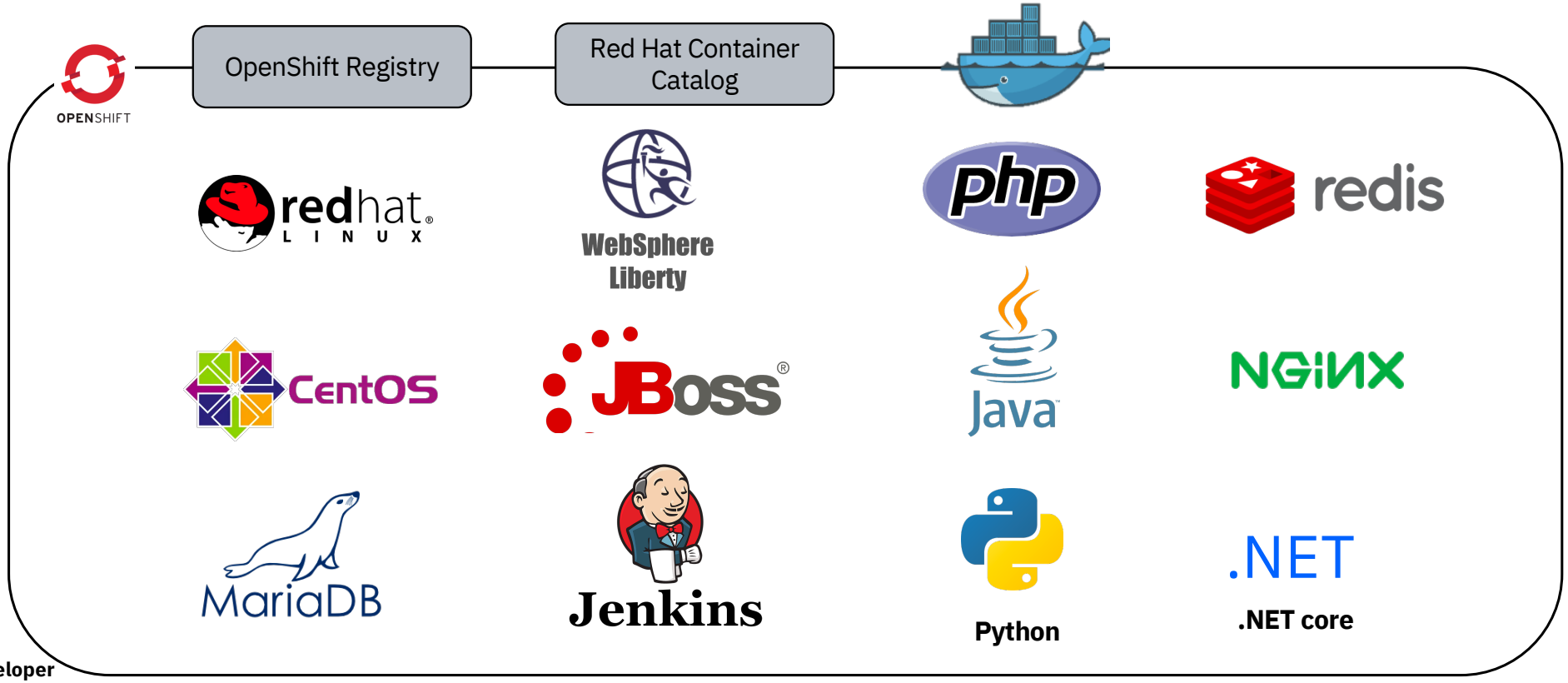
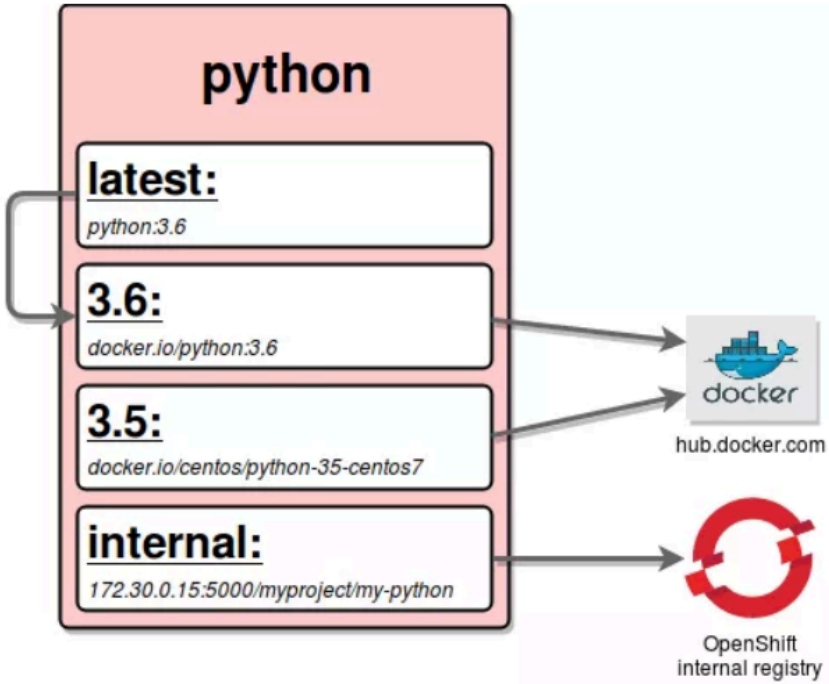


Image Streams

An image stream represents one or more Docker images identified by tags.

- Presents a single virtual view of related images
  - Can refer to images from any of the following:
    - Its own image repository in OpenShift's integrated Docker Registry
    - Other image streams
    - Docker image repositories from external registries
  - Image Streams trigger an event when underlying image is changed (even if tag remains the same)

Image Stream example



Graphic source:
https://blog.openshift.com/image-streams-faq/

Level 1: Builds

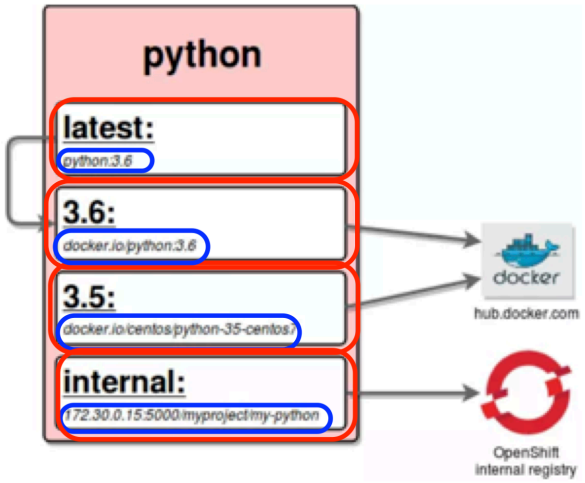
Defined via a **BuildConfig** object

- A blueprint for a process to transform base images + source code, app binaries, or Dockerfiles to an app image:
- Key attributes:
  - **Input** (input to the build process) - e.g. file, directory, GH repo etc
  - **Strategy** (how to build the app image)
    - Options:
      - *S2I* – Use a specialized builder image to generate app image (more later)
      - *Docker* - Use a Docker file to generate app image
      - *Pipeline* - A Jenkins pipeline that generates the app image from source
      - *Custom* – Encapsulate your build process via a custom builder image
  - **Output** (result of the build process) – typically an ImageStream tag

Level 2: Images

Defined via an **ImageStream** object

- An abstraction for working with Docker images inside OpenShift
- Key attributes:
  - **ImageStreamImage** (reference to actual image) - typically not used directly
  - **ImageStreamTag** (reference to a given ImageStream and tag)



Key:

ImageStreamImage

ImageStreamTag

Level 3: Deployments

Defined via a **DeploymentConfig** object

- Encapsulates the K8s Deployment and adds deployment strategies and triggers
- Key attributes:
  - **Strategy** (how to deploy if the underlying Deployment is already deployed)
    - Options:
      - *Recreate*– Blow away old deployment first and then deploy new one
      - *Rolling (default)* – Zero downtime rollout via K8s *RollingUpdate*
      - *Advanced* - (Note: require routes)
        - *Blue-Green* - running 2 versions of the deployment at the same time and moving traffic from the in-production version (the green version) to the newer version (the blue version).
        - *A/B* – partitioning requests between 2 versions of the deployment at the same time and observing the behavior
  - **Triggers** (define conditions under which the deployment is automatically triggered)

    e.g Input image updated, config changes

Level 4: Abstractions

Defines all of the additional resources needed for an app like networking, storage, security etc

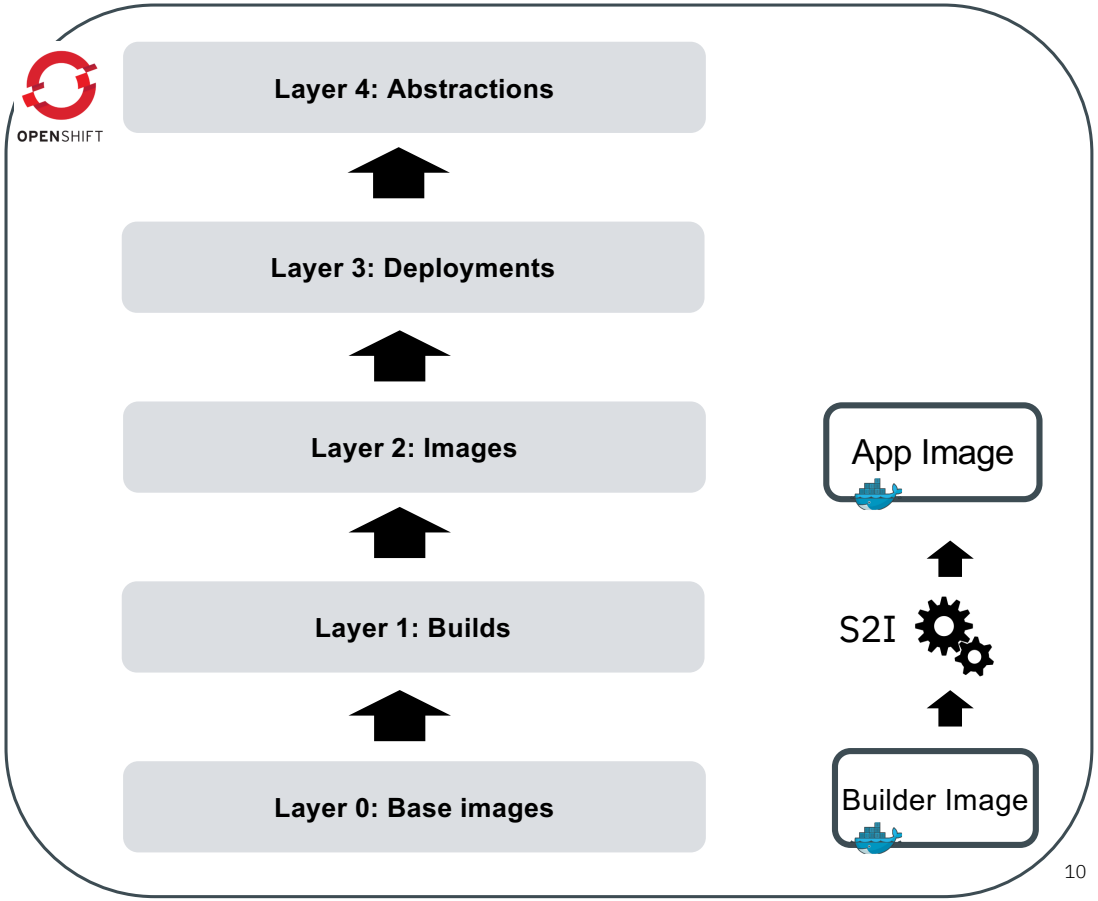
- Frequently used resources: (there are several others)
  - **Service** – Internal load balancer and router. Directs network traffic to replicated Pods
  - **Route** – Exposes a Service at a defined host name to allow access from external traffic
  - **Persistent Volume Claim** – Permanent storage used by apps to persist data after the app has stopped running
  - **Secret** – Mechanism for holding sensitive data. Decouples sensitive data from the the Pods that use them.

S2I provides a repeatable method to generate application images from source/binary code

S2I Overview

S2I is a tool that merges source code or binary into an application specific image

- Uses a builder image
  - OpenShift provides multiple builder images
    - ➢ e.g. Node.js, Java
  - Can create and add your own



Anatomy of an S2I Builder Image

Docker file contains S2I specific labels and env vars	`LABEL io.openshift.s2i.scripts-url=image:///usr/local/s2i \`
	`io.s2i.scripts-url=image:///usr/local/s2i \`
	`io.k8s.description=” S2I for Open Liberty Profile" \`
	`io.k8s.display-name="Liberty 19.0.0.4 javaProfile7" \`
	`io.openshift.expose-services="9080/tcp:http" \`
	`io.openshift.tags="runner, builder,liberty" \`
	`io.openshift.s2i.destination="/tmp"`
	
	`ENV STI_SCRIPTS_PATH="/usr/local/s2i" \`
	`S2I_DESTINATION="/tmp"`

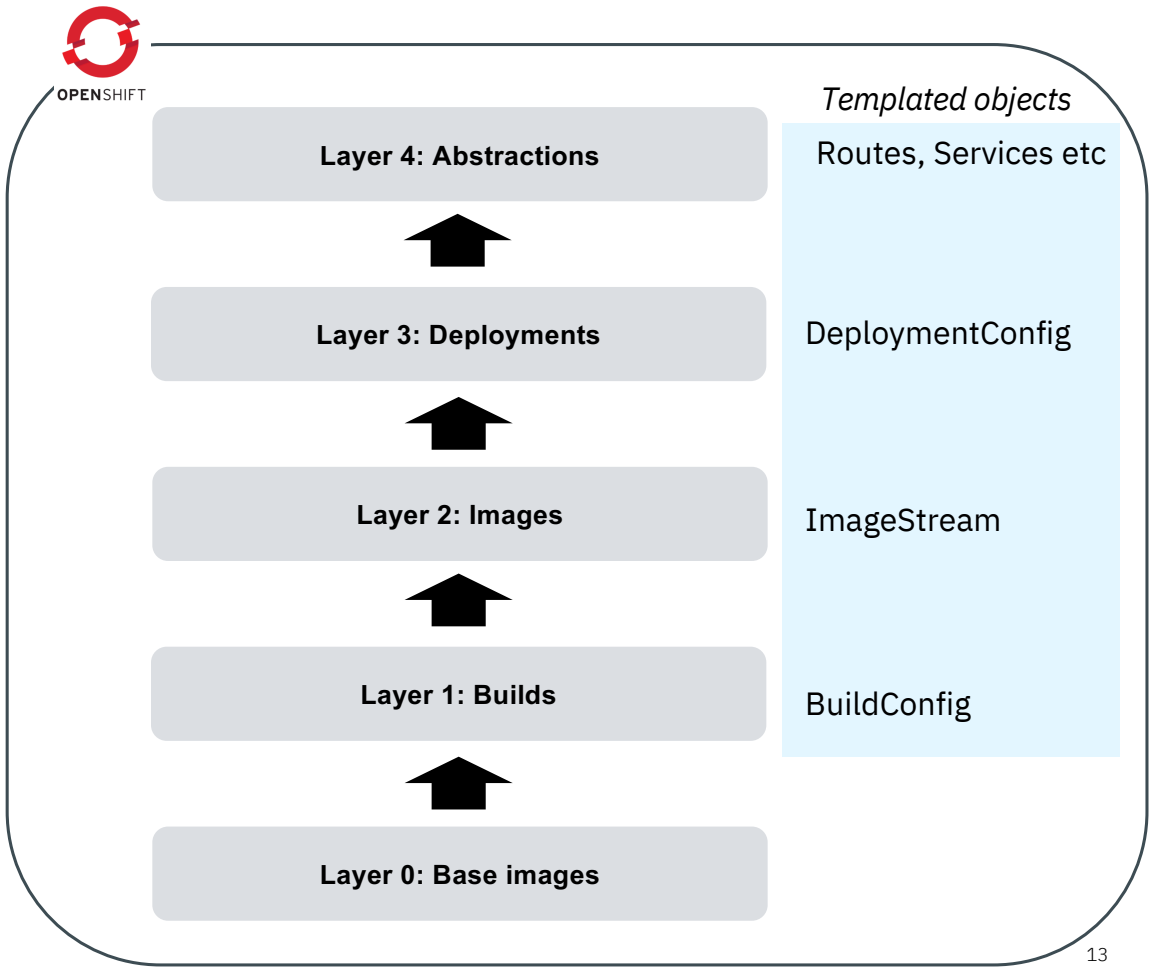
Provides base image for applications generated from the S2I image	`# This S2I image provides a base for building and`
	`# running WebSphere Liberty applications.`
	`FROM websphere-liberty:javaee7`

A series of scripts	**assemble** – Compiles and/or assembles app components from input (required) **run** – Command to run generated app image (required) **usage** – Outputs usage info about the generated image (optional) **save-artifacts** – Saves artifacts to support incremental builds (optional)

Templates provide a parameterized set of objects that can be processed by OpenShift

Templates Overview

- Templates typically contain parameterized objects for the various layers of the deployment process for an app or service
- Once created they are tightly integrated with the Web console and CLI
- Simplifies the deployment of apps or services that require several objects to be created
- OpenShift includes several templates that can be used OOTB
  - e.g. Jenkins, MariaDB etc



Anatomy of a Template

Metadata with name, description	`apiVersion: v1`
	`kind: Template`
	`metadata:`
	`  name: pbw-liberty`
	`  annotations:`
	`  openshift.io/display-name: Plants by WebSphere on Liberty w/MariaDB`
	`  description: Plants by WebSphere on Liberty App using MariaDB as the database`
	`  tags: liberty,websphere`
	`  iconClass: icon-liberty`
	`  openshift.io/provider-display-name: IBM Client Dev Advocacy.`
	`  openshift.io/documentation-url: https://github.com/djccarew/app-modernization-plants-by-websphere-jee6.git`
	`  openshift.io/support-url: https://developer.ibm.com`

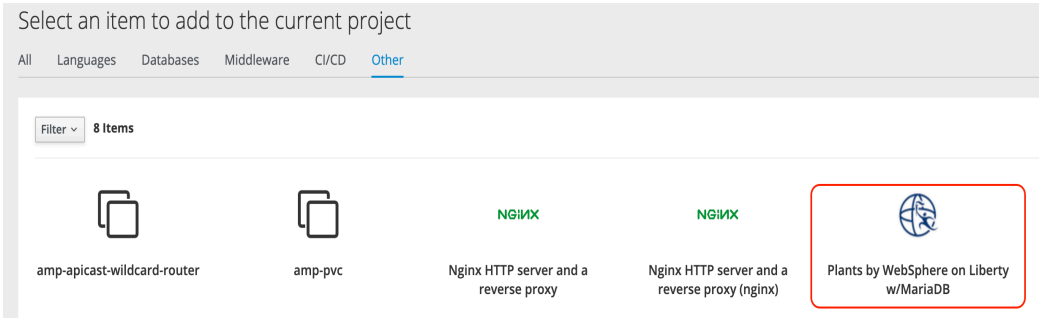
Parameters	`parameters:`
	`- name: APPLICATION_NAME`
	`  displayName: Application name`
	`  description: The name for the application.`
	`  value: pbw-liberty-mariadb`
	`  required: true`
	`...`

Parameterized objects	`- apiVersion: image.openshift.io/v1`
	`  kind: ImageStream`
	`  metadata:`
	`  name: "${APPLICATION_NAME}"`
	`...`

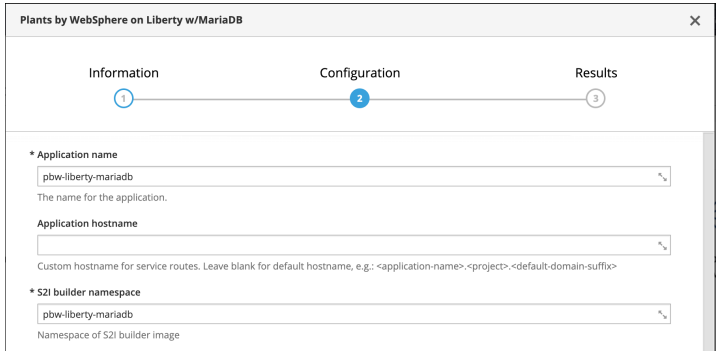
Using a template

From Web console

1 .Select template



2. Apply parameters

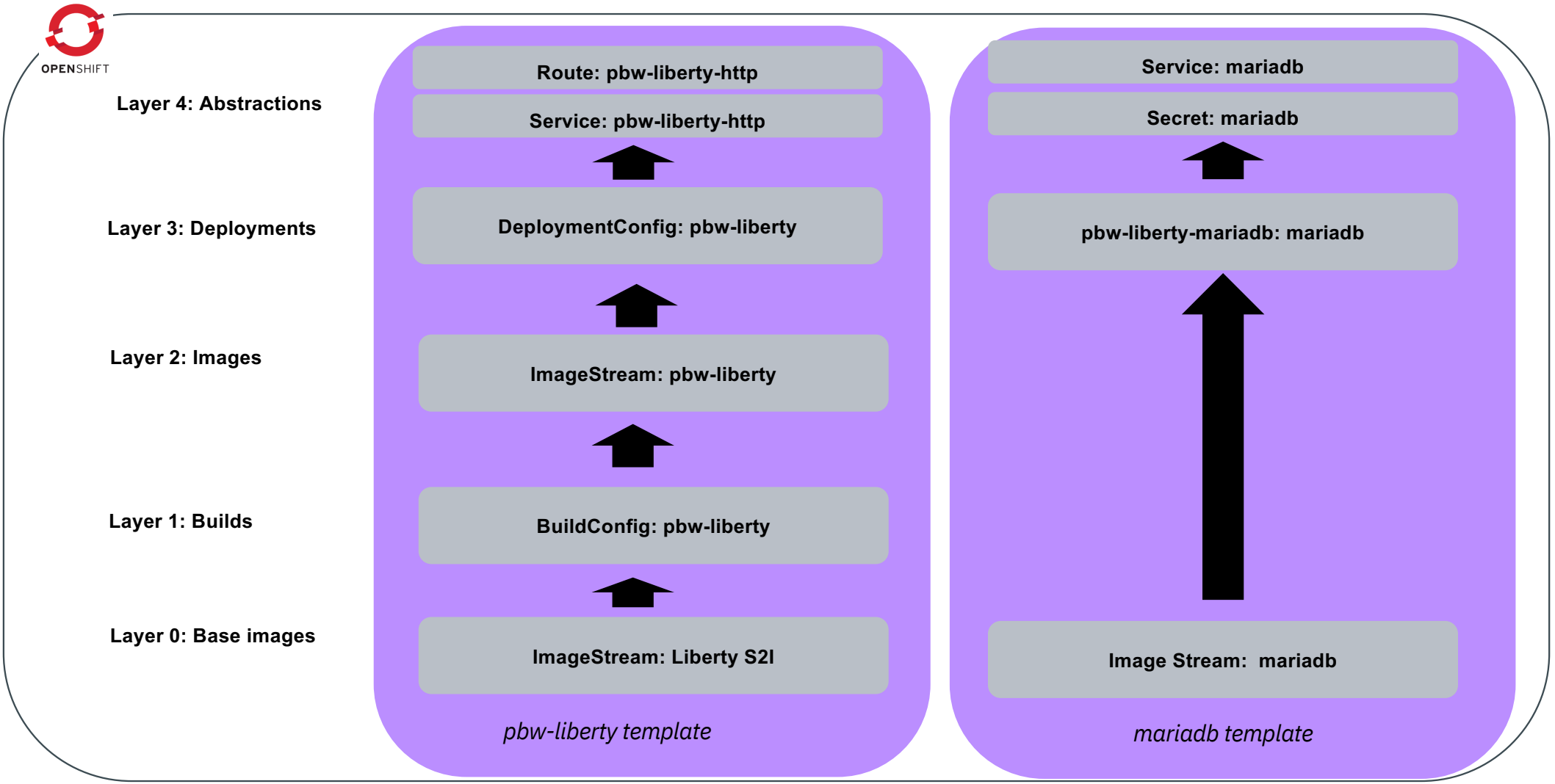


From CLI

Apply template and specify parameters by running `oc process` and then piping the result to `oc create`

```
$ oc process -f pbw-liberty \
 -p APPLICATION_HOSTNAME=foobar.com \
 | oc create -f -
```

Example – Deploy Plants by WebSphere app w/MariaDB



Resources

S2I
https://docs.openshift.com/container-platform/3.11/creating_images/s2i.html

OpenShift Templates
https://docs.openshift.com/container-platform/3.11/dev_guide/templates.html

Example Repo
https://github.com/IBMAppModernization/app-modernization-plants-by-websphere-jee6