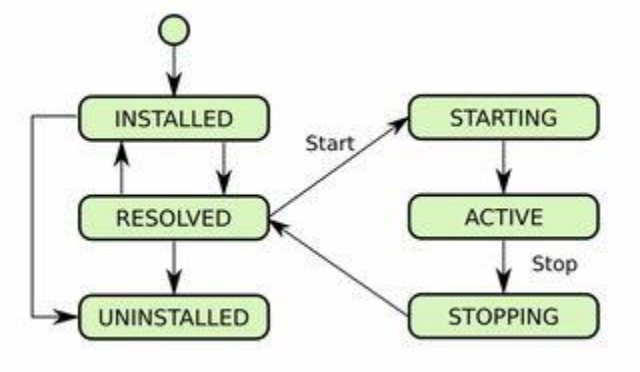
1 Difference between liferay 6.2 and dxp?

Point	DXP	6.2
Boot Strap , Aui Version	3	2
Container	OSGI	Traditional
Search	Elastic Search	Inbuilt Lucene
Single Page	Inbuilt Sena JS	Apache Lucene
Design	Clay and Lexicon	Not Support
Gogo Shell	Activate and Deactivate	Require to Remove

- Liferay Forms
- Geolocate Content & Documents
- Image and Media Selector
- Image Editor
- Webcontent diff

State of bundle



- ***Installed*** - The bundle has been successfully installed
- ***Resolved*** - All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.
- ***Starting*** - The bundle is being started. the BundleActivator.start method has been called but the start method has not yet returned.
    - When the bundle has an activation policy, the bundle will remain in the STARTING state until the bundle is activated according to its activation policy.
- ***ACTIVE***: - The bundle has been successfully activated and is running. Its Bundle Activator start method has been called and returned.
- ***STOPPING***: - The bundle is being stopped. The BundleActivator.stop method has been called but the stop method has not yet returned.

2 Types of Hook

1 Liferay DXP Login JSP Hook

- **Fragment Module Project** :- Create New Liferay Fragment Module Project - Give project name
- **Select Host OSGI bundle**
- **Search login jar + Select the jsp** file corosponding to it -bnd.bnd file will contain fragment host details

2 Struts Action Hook

- **Use -** execute your logic in particular action.
- **Many Action implemented using Struts action :-** Many Liferay portlets are implemented using Struts.
- **Help to achieve Mechanism to Override existing action : -** If you want to change the behavior of existing portlet you certainly need a mechanism to override its action class. Struts Action hook is used to override existing struts action. Struts action hook help you to achieve it.
- **Module Project:-**Click on File menu -> Click New -> Select Liferay Module Project and provide project name as StrutsActionHook and click next Provide class and package name
- **Can find url from struts-config.xml.**
    - tomcat server ? liferay-ce-portal-7.0-ga3 ? tomcat-8.0.32 ? webapps ? ROOT ? WEB-INF ? struts-config.xml.
- **While calling action**- e.g. Update email , update terms and condition
- **StrutsAction Service :-** struts actions can be overridden directly with service = StrutsAction.class and only struts action defined in struts-config.xml can only overridden

```
@Component(
 immediate=true,
 property={
 "path=/portal/update_terms_of_use"
 },
 service = StrutsAction.class
)
public class CustomTermsOfUseAction extends BaseStrutsAction {

 @Override
 public String execute(StrutsAction originalStrutsAction, HttpServletRequest request,
HttpServletResponse response)
 throws Exception {
  System.out.println("Calling Custom Termsof Use Action");
```

```
 //you logic goes here
 return super.execute(originalStrutsAction, request, response);
}
```

Filter Hook – Link

- **Intercepting the request before code execute :-** Filter hook are used for intercepting HTTP request to execute piece of code before it reaches to execution logic. This is like HTTP Filter.
- create a Liferay Module Project.
- Provide component name and package path for the filter class

```
@Component(
 immediate = true,
 property = {
 "servlet-context-name=",
 "servlet-filter-name=Custom Filter",
 "url-pattern=/*"
 },
 service = Filter.class
 )
public class CustomFilter extends BaseFilter {
 @Override
 protected void processFilter
```

Language properties Hook

Change Liferay Portal's messages and labels :-A language properties hook lets you change Liferay Portal's messages and labels to suit your needs.

```
@Component(
```

```
 property = { "language.id=en_US" },
 service = ResourceBundle.class
)
public class CustomLanguageComponent extends ResourceBundle {
```

Model Listener hook

Model Listener

- Model Listener is used to listen to Liferay Model Events.
- We can perform custom action on Before/After the persistence event of Model objects using Model Listener.
- Model Listeners implement ModelListener interface.

```
@Component(
immediate = true,
service = ModelListener.class
)
public class MyCustomRoleModelListenerPortlet extends BaseModelListener<Role> {
```

Service Wrapper Hook

- **portal service customization :-**We can achieve portal service customization with the help of Service Wrapper hook.
- Basically, Liferay generates wrapper classes for all services. For example, BlogsEntityLocalServiceWrapper class is created for BlogsEntityService which contains methods to add, update, delete Blogs entry.
- If we want to modify any of these services then we need to create custom class that extends BlogsEntityLocalServiceWrapper via Service Wrapper hook.
- Once we deploy such hook, Liferay portal container will use custom service class instead of original service wrapper class.
- We can achieve portal service customization with the help of Service Wrapper hook.
- servicewrapper” template.
- Now select service for Blogs entry
- Override addEntry method of BlogsEntryServiceWrapper class

```
@Component(

immediate = true,
```

property = {},
service = ServiceWrapper.class
)
public class CustomBlogsEntryServiceWrapper extends BlogsEntryLocalServiceWrapper
Service Hook

- We can achieve portal service customization with the help of Service Wrapper hook.

Indexer Hook

- Use - The main purpose of the Indexer Hook is to execute our custom code which will be invoked after the default Indexer class has done its work.
- Job Title - of user will not able to search
- Using Indexer Post Processor hook

```
@Component(
   immediate = true,
   property = {
      "indexer.class.name=com.liferay.portal.kernel.model.User",
      "indexer.class.name=com.liferay.portal.kernel.model.UserGroup"
   },
   service = IndexerPostProcessor.class
)
public class UserEntityIndexerPostProcessor implements IndexerPostProcessor
```

3 Steps for the dxp upgrade

Data Clean-up:-

- **Remove unused** sites , Instances , web content, Users, layout , Other Data , Duplicate Structure Name

Liferay Database Migration

- **Database Migration Tool** **:-** Liferay Provide Database Migration Tool
- **Disable Search Index**
- **Backup of DB** :- Take Backup of DB
- Create and Import DB :- Create New DB and **Import DB**
- portal-setup-wizard.properties Provide the db connection properties
- Copy Data Folder
- Disable indexer and autoUpgrade
- Run Tools portal-tools-db-upgrade-client in [Liferay Home]/tools
- Upgrade properties files
    - app.server.properties

    - database.properties
    - portal-setup-wizard.properties
- Verify BuildNumber select * from release_ where servletcontet = 'portal'
- Double click on “db_upgrade.bat” file
- Enable the Search Index

Code Migration :-

- **Code Upgrade Tool :-** Liferay provide code upgrade tool
- JSP hook take reference of files and create
- Portlet Migration code upgrade too
- Code Upgrade Tool in Liferay Developer Studio (Project -> Liferay Code Upgrade Tool menu).
- After importing project we can click “Find breaking changes” button,
- Changes require to do and corrosponding jira li

JSP Hook

- Good to create from scratch
- Find diff and apply
- LayOut Template
- Create LayOut Template

Theme

- Short term solution
- Backup
- yo liferay-theme:import
- gulp upgrade
- Updates the theme’s Liferay version
- Updates the CSS
- Update Template
- Suggests specific code updates
- gulp upgrade

- Elastic Search

Elasticsearch is a search engine based on the Lucene library
. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. ... According to the DB-Engines ranking, Elasticsearch is the most popular enterprise search engine.

Some of its primary features include distributed full-text distributed search, high availability, powerful query DSL, multitenancy, Geo Search, and horizontal scaling.

4 Customize search liferay

- Gsearch
- PostIndexer
- -https://help.liferay.com/hc/en-us/articles/360018161651-Customizing-Liferay-Search
- JSP Hook to overide look and feel

Service Builder

10 Portlet Filter

- Action Filter : To Intercept only The Action Requests
- Render Filter : To Intercept only The Render Requests
- Resource Filter : To Intercept only The Resource Requests
- Base Filter : To Intercept all the http request comes to your portlet.
- Create A Module Project for Portlet

```
import org.osgi.service.component.annotations.Component;
import javax.portlet.filter.PortletFilter;
@Component(immediate = true,
 property = {
  "javax.portlet.name="+MyPortletKeys.My,
 },
 service = PortletFilter.class
)

public void init();
public void destroy();
public void doFilter();
public class EncodingPersonEmailsRenderFilter implements RenderFilter
```

11 JSR 286 Standard

- IPC

- Serve Resource
- Portlet filter

12 MVC Command Render

- If your render logic is complex or you want clean separation between render paths, use MVCRenderCommands.
- Each render URL in your portlet's JSPs invokes an appropriate render command class.

```
<portlet:renderURL var="bladeRender">
        <portlet:param name="mvcRenderCommandName" value="/blade/render" />
</portlet:renderURL>
@Component(
 property = {
 "javax.portlet.name=org_javasavvy_web_leave_portlet",
 "mvc.command.name=/blade/render"
 }, service = MVCRenderCommand.class
 )
public class ViewLeaveInfoCmd implements MVCRenderCommand{
```

13 How to connect multiple data source liferay? link

Open your portal-ext.properties and enter the database properties

- jdbc.ext.driverClassName=com.mysql.jdbc.Driver
- jdbc.ext.username=root
- jdbc.ext.password=root
- jdbc.ext.url=jdbc:mysql://localhost/datasourcetesting?characterEncoding=UTF-8&dontTrackOpenResources=true&holdResultsOpenOverStatementClose=true&serverTimezone=GMT&useFastDateParsing=false&useUnicode=true

Create ext-spring.xml file in service builder

- src/main/resources/META-INF/spring/ext-spring.xml

```
 <bean  class="com.liferay.portal.dao.jdbc.spring.DataSourceFactoryBean"
id="liferayDataSourceFactory" >
        <property name="propertyPrefix" value="jdbc.ext." />
    </bean>
    <bean
class="org.springframework.jdbc.datasource.LazyConnectionDataSourceProxy"
id="liferayDataSource" >
        <property name="targetDataSource" ref="liferayDataSourceFactory" />
    </bean>
```

Provide data source in service.xml file

- ```
  <entity name="Employee" local-service="true" remote-service="false"
  table="employee" data-source="secondDatabase">
        <column name="eid" primary="true" type="int"></column>
        <column name="ename" type="String"></column>
   </entity>
  ```

Add spring dependency

compileOnly group: "com.liferay", name: "com.liferay.portal.spring.extender.api", version: "4.0.0"
  ```

14 How to create custom workflow? Custom Entity Workflow

- ***Make Entity As Asset***
- ***Add workflow related column in service.xml***
- ***Register workflow handler***
- ***Register AssetRenderer with Entity***
- ***Update ServiceImpl with updatestatus , updateasset,startworkflowinstancemethod***
- ***Map Entity with workflow definition in control panel***

- **Make Entity As Asset :-** Make the Leave entity as Asset Type
- **Add workflow related column in service.**xml
  - Add workflow related columns in service.xml for Leave entity
- **Register workflow Handler**
  - Register workflow Handler with Leave entity
- **Register AssetRenderer with Leave entity**
  - Register AssetRenderer with Leave entity to make Leave entity visible in Workflow Configuration Screen
- Update ServiceImpl with updatestatus , updateasset , WorkflowHandlerRegistryUtil.startWorkflowInstance()
  - Update LeaveLocalServiceImpl with below:
  - Add updateStatus() method, which will be called by Workflow handler at runtime

    - invoke assetLocalService.updateAsset() method to add entry to asset_entry table
    - Invoke WorkflowHandlerRegistryUtil.startWorkflowInstance() method to start workflow
- **Map Entity with workflow definition in control panel**
    - Map the Leave entity with Workflow definition in the control panel

How to implement action command?MVCACTIONCOMMAND

- Liferay's MVC Portlet framework enables you to handle MVCPortlet actions in separate classes.
- This facilitates managing action logic in portlets that have many actions.
- Each action URL in your portlet's JSPs invokes an appropriate action command class.
- <Liferay-portlet:actionURL name="editleave" var="editleaveURL"/>

```
@Component(
 property = {
 "javax.portlet.name=org_javasavvy_web_leave_portlet",
 "mvc.command.name=leave_editLeave"
 },
 service = MVCActionCommand.class
 )
public class EditLeaveActionCommand extends BaseMVCActionCommand {
```

Liferay DXP ActionCommand Hook

- ***Purpose :-*** tutorial will drive you on customizing liferay portlet actions such as user authentication action,edit journal article action, edit blog action etc.

```
@Component(
 immediate = true,
 property = {
 "javax.portlet.name=com_liferay_login_web_portlet_LoginPortlet",
 "mvc.command.name=/login/login",
 "service.ranking:Integer=100"
 },
 service = MVCActionCommand.class
 )
public class CustomLoginActionCommand extends BaseMVCActionCommand {
 @Override
 protected void doProcessAction(ActionRequest actionRequest, ActionResponse
actionResponse) throws Exception {
```

Modal Hints

Purpose - In Liferay you can easily modify DB column size.

- Proccess - src/META-INF -> portlet-model-hints.xml
  <field name="columnName" type="String">
  <hint name="max-length">4000</hint>
  </field>

- Here is the example for Column with type String. By default if we use type as a string in service.xml then size will be 75.
- By doing few changes inside portlet-model-hints.xml. This configuration file is available inside src/META-INF directory. To increase column size do something like this.
- If max-length is less-then 4000 then column type will be Varchar.
- If max-length is equal to 4000 then column type will be STRING. If max-length is greater then 4000 then column type will be TEXT.

How to check if module have issue

- Check logs
- Open Gogo Shell lb command
- diag bundleid

Service Context

- The ServiceContext class holds contextual information for a service.
- It aggregates information necessary for features used throughout Liferay's portlets, such as permissions, tagging, categorization, and more.

24 Patching Tool

- [Liferay Home]/patching-tool/patches
- patching-tool info
- patching-tool install
- To make sure the all changed OSGi bundles replace the existing ones, it is recommended to delete the osgi/state folder from the Liferay Home folder.
- patching-tool index-info

- database.indexes.update.on.startup=true

Self Register page which hook require to create?

- Login JSP Hook Require to create

OSGI benefits

- You can install, uninstall, start, and stop different modules of your application dynamically without restarting the container.
- Your application can have more than one version of a particular module running at the same time.
- OSGi provides very good infrastructure for developing service-oriented applications, as well as embedded, mobile, and rich internet apps.

Liferay Clustering

Step 1. Ensure that ports are open

For Liferay clustering to be successful, the hosts need to be able to communicate with each other to send clustering packets across the network.

TCP Unicast clustering in Liferay

Step 2. Configure TCP Unicast

Create a TCP Unicast file

For the next step, an XML file (for simplicity, we're naming it dxp-clustering.xml) containing the TCP Unicast configuration should be created. Please note that we recommend the file be copied to the global library filesystem location

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="urn:org:jgroups"
      xsi:schemaLocation="urn:org:jgroups http://www.jgroups.org/schema/jgroups.xsd">
   <TCP bind_port="7800"
        recv_buf_size="${tcp.recv_buf_size:5M}"
        send_buf_size="${tcp.send_buf_size:640K}"
        max_bundle_size="64K"
        sock_conn_timeout="300"

        thread_pool.enabled="true"
        thread_pool.min_threads="0"
        thread_pool.max_threads="20"
```

```
        thread_pool.keep_alive_time="30000"

        />

<!--       <TCPPING async_discovery="true"
           initial_hosts="${jgroups.tcpping.initial_hosts:localhost[7800],localhost[7801]}"
           port_range="2"/>
-->
<!-- <JDBC_PING
connection_url="jdbc:mysql://[DATABASE_IP]/[DATABASE_NAME]?useUnicode=true&characterEncoding=UTF-8&useFastDateParsing=false"
   connection_username="[DATABASE_USER]"
   connection_password="[DATABASE_PASSWORD]"
   connection_driver="com.mysql.jdbc.Driver"/> -->

<!--  <JDBC_PING
  datasource_jndi_name="java:comp/env/jdbc/LiferayPool"/> -->



<!--          <S3_PING location="$name_of_the_s3bucket"
                                    access_key="$AWS_access_key"
                                    secret_access_key="$AWS_secret_key"
                                    timeout="2000"
                                    num_initial_members="2"/> -->


   <FILE_PING location="/liferay/shared/document/library/jgroups" />

     <MERGE3  min_interval="10000"
         max_interval="30000"/>
   <FD_SOCK/>
   <FD_ALL timeout="9000" interval="3000" />
   <VERIFY_SUSPECT timeout="1500"  />
   <BARRIER />
   <pbcast.NAKACK2 use_mcast_xmit="false"
            discard_delivered_msgs="true"/>
   <UNICAST3 />
   <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
            max_bytes="4M"/>
   <pbcast.GMS print_local_addr="true" join_timeout="3000"
           view_bundling="true"/>
   <UFC max_credits="2M"
      min_threshold="0.4"/>
   <MFC max_credits="2M"
      min_threshold="0.4"/>
   <FRAG2 frag_size="60K"  />
   <!--RSVP resend_interval="2000" timeout="10000"/-->
   <pbcast.STATE_TRANSFER/>
</config>
```

3 Add properties files

cluster.link.enabled=true
cluster.link.autodetect.address=database_host:port
cluster.link.channel.properties.control=dxp-clustering.xml
cluster.link.channel.properties.transport.0=dxp-clustering.xml

Enabling Cluster Link automatically activates distributed caching.

cluster.link.enabled=true

When you enable Cluster Link, Liferay DXP's default clustering configuration is enabled. This configuration defines IP multicast over UDP. Liferay DXP uses two groups of channels from JGroups to implement this: a control group and a transport group. If you want to customize the channel properties, you can do so in portal-ext.properties:

cluster.link.channel.name.control = liferay-channel-control

cluster.link.channel.properties.control tcp-unicast-62.xml

cluster.link.autodetect.address abc.com:9000

-

Remaining Questions

- How to solve dependency problem liferay
- How to solve dependency problem liferay?
- Use of export and import
- how to provide multiple version? Elastic Container Link