**MDW Cloud Mode Considerations**

MDW applications deployed in Cloud Mode consist of business logic that's developed entirely in the form of workflow assets. Examples of these assets include Processes, Rules, Pages, Scripts, Templates, Reports, Camel Routes, Dynamic Java, and even Jar archives. MDW supports the ability for clients to collaborate in developing workflow processes and assets without needing to install any local infrastructure. To distinguish non-cloud mode we refer to this as Dedicated Mode. It's important to note that this doesn't mean that dedicated hardware is provisioned for non-cloud deployments (which are almost always hosted in a virtualized environment). The distinction is more accurately understood as avoiding the need for a custom application deployable (such as an EAR file, WAR file, or OSGi bundle). Cloud-based projects still allow users to deploy custom Java code, but without any application deployment in the traditional sense. Access controls are rigorously enforced through role-based authorization, with versioning, rollback and strict audit logging to ensure accountability. To insulate workflow apps from others hosted in the same environment, configuration settings for a cloud-based project apply at the workflow package level.

To help guide the decision about which is the right development model for a particular situation, the table below summarizes key differences between cloud and dedicated modes. In general, if your project involves an extensive existing codebase or needs direct access to container resources, you should consider choosing the dedicated deployment model; otherwise you may prefer the simplicity and rapid startup afforded by the cloud model. Either way, this decision is strictly a design-time issue. At runtime workflows are processed by the MDW engine in exactly the same way regardless of the model used to create them.

|  |  |  |
| --- | --- | --- |
| **Cloud Versus Dedicated** | **MDW BPM Deployment** | |
| **Cloud Mode** | **Dedicated Mode** |
| **Custom Code** | Through Dynamic Java all aspects of MDW extensibility are supported. This includes Activity Implementors, Event Handlers. Registered services such as Scheduled Jobs are configured through Java Annotations.  Application and third-party code can also be deployed as Jars assets. | Client app maintains its own deployables, such as EAR, WAR, or OSGi bundles. MDW features are accessed through libraries and the Service Registry. |
| **Web Pages** | Cloud-based Pages with custom Managed Beans. Web resources such as stylesheets and javascript are stored as workflow assets. Can encompass an entire custom WebApp. | MDW web components and features are embedded in a client app's WAR archive. Reusable templates, stylesheets and scripts are directly embedded in the client WAR. |
| **Configuration** | Settings apply at workflow package level (with global settings as fallback). | Global MDW settings are customized. |
| **Auth. Control** | Package visibility limited by group membership. Assets strictly controlled by role-base authorization and audit logging. | Assets governed as in Cloud Mode. Changes to deployables require file system access as well. |
| **Container Resources** | Standard MDW JDBC DataSource and JMS Queues. | Custom application DataSource and JMS Queues for external access. Custom JMS broker config. |
| **Web Services** | SOAP and REST services configured through standard MDW context root, or through Camel route assets. | Application-managed web contexts, with the option of custom listeners for exotic use cases. |
| **TIBCO RVD** | Listener limited to a single Bus URI. Adapters can consume any number of external services. | Supports dedicated BusResponders.xml configuration (needed for hosting multiple URIs and for certified messaging). |
| **Testing** | Dynamic Java JUnit tests, and MDW function/Cucumber/load test facility. | Other, non-standard unit and regression test frameworks. |
| **Debugging** | Developers can create a local cloud project running on Tomcat or push to Cloud Foundry. | Local, dedicated debug sessions |