Migrating Java Apps to Apprenda

A Prescriptive Guide

Cloud Enablement

Apprenda's Platform subsystems automatically assist applications in being "Cloud-ready" simply by virtue of their deployment by the Platform. Things like workload scalability, communications bus, and more are native parts of the Platform that applications sit "atop" and automatically leverage. That said, there are design patterns and architecture decisions that Developers can make during development of an app that take full advantage of the Platform's capabilities. These approaches should also be considered when migrating an app portfolio to Apprenda. This guide offers prescriptive instructions on the most common considerations that should be made while migrating a Java Web Application to Apprenda. We recommend you read this document in its entirety and build a migration plan employing these "recipes" for each application you are migrating.

The document starts with common Cloud-readiness considerations, and progressively moves through leveraging Apprenda for enhanced Cloud-readiness and eventually pure SaaS.

While reviewing each task or recommendation in this guide, keep in mind these icons:

☁ !	Required for Cloud readiness
a !	Required for Apprenda readiness
a ☁ ~	Optional for Apprenda readiness

Application Structure

Packaging and container dependencies

It is common practice with Java applications to package any UI and Service tiers into a single WAR file. Apprenda requires this method of packaging; components packaged into the WAR file will then be deployed and scaled as a single tier on the Platform. In Apprenda terminology, this is called the **Java Web Application tier**. Apprenda supports the deployment of Java Web Application components to the Java web containers listed in this section of our documentation. Applications that have not built a dependency on any container-specific functionality can be deployed as-is on Apprenda with little or no changes at all. Some (although not all) dependencies on supported containers can be retained; please contact support@apprenda.com to discuss the types of dependencies that are supported.

Additional changes required will depend on the coupling that has been made with the server, OS, and other infrastructure components, as Apprenda handles much of the infrastructure management, scaling, and deploying out of the box. Apprenda's documentation provides more information on the deployment mechanics for the Java Web Application tier, as well as descriptions on the different tiers supported in an Apprenda application.

Further Reading
The Java Web Application tier:
http://docs.apprenda.com/current/java-tier

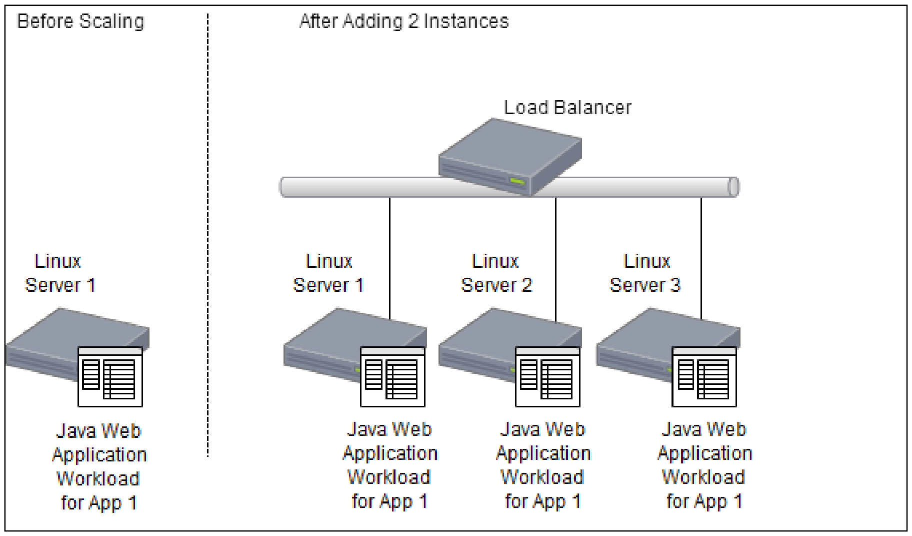

Before Scaling
After Adding 2 Instances
Load Balancer
Linux Server 1
Java Web Application Workload for App 1
Linux Server 1
Linux Server 2
Linux Server 3
Java Web Application Workload for App 1
Java Web Application Workload for App 1
Java Web Application Workload for App 1

Figure 1: An application scaling at the Java Web Application tier (with automated load balancing by Apprenda software load balancer)

Database Connections and Tokenization

Database connection strings, resource locations and service endpoints should all be specified in configuration.

Part of Apprenda's dynamic configuration engine is a tokenization system. **Database connection strings, resource locations and service endpoints should all be specified in configuration** and tokenized for dynamic deployment on Apprenda resources. http://docs.apprenda.com/current/app-config-tokens.

It should be noted that if the application already abstracts those configuration concerns by performing JNDI lookups, for instance, it may not be necessary to change it at all since Apprenda can configure a JNDI context at deploy time with the same dynamically replaced tokens mentioned above.

Remove all local machine dependencies.

Network and other configuration details must be machine independent in order for applications to be easily scalable and available. Apprenda manages the details of the hardware and network infrastructure so the developer doesn't have to. For an overview of the functional nature of this feature visit http://docs.apprenda.com/current/apprenda-deployment. Cloud-readiness means that any dependencies on a specific operating system construct, like a filesystem or the server name, should be removed.

Common Pitfalls

In-memory State: Using the Apprenda Platform gives Developers the advantage of not needing to know the cloud infrastructure details. This feature may require a change to how an application accesses state information. As a result, avoid tying your application to a specific network location or machine instance managed by the Platform. It may work but it will likely misbehave.

If an application currently saves user state to a Servlet HTTPSession and adheres closely to those APIs, it may be able to transparently leverage Apprenda's distributed Session store without any code changes. For high-traffic apps, another option is to create a stateless application by creating a lightweight secure session token on the client side (cookie) and using a distributed caching strategy on the server side to maintain application state.

Authentication: Applications which previously had components to authenticate users should now rely on the Apprenda Platform for authentication. Using Platform authentication enables SSO for all applications according the needs and requirements of the enterprise. Visit the section entitled "Apprenda's Authentication Gateway" in our Platform documentation here http://docs.apprenda.com/current/java-tier.

When an application defers authentication to Apprenda's mechanism, a User accessing the application is presumed to be authenticated against the Platform, so the need to have multiple logic paths (for anonymous vs. logged in) can be reduced to just the logged-in path.

Leveraging the Apprenda Platform

So far, this document has covered common topics that Developers should consider *about their existing code* when migrating an app to the Apprenda Platform.

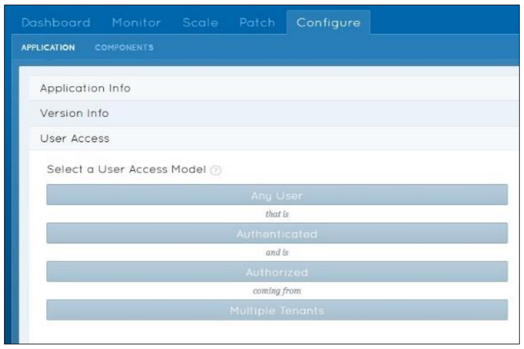
The Platform offers a wide arrange of functionality that guest applications can take advantage of simply by running on the Platform and more advanced capabilities by tapping into the different runtime and service-based APIs available.

The following sections outline some key aspects that should be considered when moving to Apprenda to enhance existing applications.

Wiring-In Platform Authentication

The Platform, as mentioned above, offers authentication capabilities to guest applications. There are a few ways to take advantage of this capability depending on the level of customization and control desired. Enabling authentication on the Platform is as simple as turning on the functionality through the Configure tab in the Developer Portal UX's application overview or the DeploymentManifest.xml file. The following resources explain both approaches (http://docs.apprenda.com/current/app-level-redesign#useraccess, http://docs.apprenda.com/current/deployment-manifest#settings). To allow the Platform to specify authentication and identity information for you application, remove all logic in the application that accepts credentials and initiates identity information (storing username, etc. in session state). Instead, access user identity information via the Apprenda User runtime API. This context will be populated for you by virtue of the fact that the Platform has authenticated the User prior to their access to the app.

Two options for enabling authentication at the Platform level for your app:

Application Settings in the Developer Portal	DeploymentManifest.XML
 Dashboard Monitor Scale Patch Configure APPLICATION COMPONENTS Application Info Version Info User Access Select a User Access Model Any User that is Authenticated and is Authorized coming from Multiple Tenants http://docs.apprenda.com/current/ app-level-redesign#useraccess	`<applicationServices level="Authentication" />` http://docs.apprenda.com/current/deployment-manifest#settings

Centralized Logging

One of the most important consideration when developing cloud applications is log management. As the footprint of your application grows, it becomes more difficult to track where errors occur. Apprenda offers a logging API that facilitates centralized logging from disparate application workloads. These log messages are stored centrally along with contextual information about the application, User, Tenant (if applicable) and machine on which the event occurred. Developers can then manage their logs through the Developer Portal and Platform Operators can monitor all logs across the platform. For more information on the logging interface, please refer to: http://docs.apprenda.com/current/logging.

Remove the use of file system logging (local machine dependency) and replace it with Apprenda's distributed logging mechanism for a centralized view of events in disparate application workloads.

Apprenda handles logging for Java applications by leveraging log4j. In order to take advantage of Apprenda's centralized logging service, Java Developers need only implement log4j in their code as they normally would.

```
Writing to the Apprenda logs:

    private static Logger Logger = LogManager.getLogger("MyService");
    ...
        if (Logger.isDebugEnabled()) {
            Logger.debug("Logging in user " +
                        currUser.getFirstName() +
                        " with ID" + currUser.getId());
        }


Notice that we do not need to specify meta information about the log event such as the application name
or source.  This information is inferred by Apprenda as the event is logged.
```

Leveraging Apprenda for SaaS

Apprenda allows Developers to instantaneously convert a plain Java application into a SaaS offering with just a click of a button. By using our APIs, Developers can focus on the core functionality of their application, while letting Apprenda orchestrate such constructs as app-level multi-tenancy, data modeling, authorization, and metering. The following section will demonstrate how to leverage the Apprenda APIs to satisfy common SaaS requirements.

Apprenda Runtime Contexts

Apprenda surrounds all Platform requests with various contextual information. During your code's execution on the Platform, these contexts contain additional information such as Tenant-specific connection strings, the name of the User that owns the executing thread, their permissions within the scope of your app, and more.

Detailed information about these runtime context for Java applications can be found here http://docs.apprenda.com/current/wars#contexts.

Securing Your Application via Roles

Apprenda manages and maintains Roles and Role membership for a Tenant and its Users at the Platform level. The Account Administrator (or someone with appropriate permission) can define permissions to secured functionality via Apprenda's Role system on a per-application basis.

To take advantage of this capability, an application must have at least Authorization enabled as its Application Service level and include Securables in its definition. Developers use the Apprenda Guest App API to inquire whether or not a User (via their Role membership) can access certain application functionality. The Tenant can define permissions by specifying which Role membership is required to use a certain application functionality denoted by the Securable name and description.

A securable can either be defined statically in application configuration (Static Securable), or created at runtime (Runtime Securable). The following example shows how to get an Iterable of all securables from either type.

```
GuestAppContext guestCtx = ApprendaGuestApp.getContext();
if (guestCtx.isAuthorizationEnabled()) {
    AppVersion appVersion = guestCtx.getAppVersion();
    Iterable<Securable> runtimeSecurables = appVersion.getRuntimeSecurables();

    Iterable<Securable> staticSecurables = appVersion.getStaticSecurables();
}
```

Once the application has access to a Securable (Static or Dynamic), it can check if the User has access to the Securable. Further information can be found in the API docs for AppVersion.

```
// check if the current user has access to the "Manage Users" securable
if (guestCtx.isAuthorizationEnabled()) {
    AppVersion appVersion = guestCtx.getAppVersion();
    // check if the current user has access to the "Manage Users" static securable
    appVersion.hasAccessToStatic("Manage Users");
    // check if the user has access to the dynamic securable "SecretDocument 5.0.0"
    appVersion.hasAccessToRuntime("SecretDocument 5.0.0");
}
```

The application can retrieve the set of Roles of the Tenant and the set of Roles for a given User. The application is also able to search for Users in the given Tenant based on the fields and the search string.

```
User currUser = guestCtx.getUser();
Tenant currTenant = guestCtx.getTenant();

// these are all roles defined for the tenant
Iterable<Role> tenantRoles = currTenant.getTenantRoles();

// let's see the roles of the current user
Iterable<Role> userRoles = currTenant.getUserRoles(currUser.getId());

// let's get the first  many gmail users we have
String searchString = "gmail.com";
List<UserSearchColumn> searchFields = new  ArrayList<UserSearchColumn>();
searchFields .add(UserSearchColumn.Email);
PagedResult<? extends User> userSearchPage =
        currTenant.searchUsers(searchString , searchFields , new PagingParams(1,10));
// now, iterate over the list of users and look at their names
User firstUser = userSearchPage.iterator().next();
```

Metering Basics Using the Apprenda API

Developers can include programmatic API references to instruct the Apprenda Platform to track the utilization of specific functionality within an application. The Platform will match the 'Feature' marked in the implementation of the functionality with a component specified in the Subscription assigned to the user. For details, please see: http://docs.apprenda.com/current/features and http://docs.apprenda.com/current/features-and-editions.

There are four types of features that can be tagged:

- Toggle
- Boundary
- Block
- Limiter

The following example demonstrates how to utilize a limiter feature called 'Projects' and apply it to functionality – in this case a method on a Java Web Service.

```
GuestAppContext guestCtx = ApprendaGuestApp.getContext();
public void CreateProject(Project project)
{
        LimitMeter limitMeter = guestCtx.getMeters().getLimit("projects");

        if (!limitMeter.getState().isExhausted()) {
                ProjectRepository.getInstance().save(project);
        }
}
```