# Release management with MTA

With the multitenant model, the package typically only needs to be installed on a single server. This greatly simplifies the release management process.

At the same time, multitenancy increases the risk of applying a new release version and the effects of that. As there is a single software instance serving multiple tenants, an update on this instance may cause downtime for all tenants even if the update is requested and useful for only one tenant.

Hence right from the start, each tenant system will be considered to be an implementation of base interfaces, with the underlying datastore at individual level.

The datastore will have all objects designed at schema level, ensuring logical partitioning. If all tenants accept same logic without any customization, they can all co-exist within the same application and database domain.

The scalability of a tenant will be horizontal or vertical as the case may demand. The design allows for movement of entire datastore to a separate deployment unit. This can be a separate instance, server or on cloud.

## Change Management Concerns

Each tenant will demand specific branding “always” and system has to support any configuration required for this.

Any feature request as part of tenant’s own business process will need to be treated as individual and deployed to that tenant’s area without affecting on other tenant system.

Any bugs or general changes in the system (required by FIS) will need to be provided to “all” the tenants as part of common system update. The tenant can choose to delay the deployment at individual level, however up-to a given threshold for checking the stability of the platform and compatibility with their own business processes.

Any change mandated by regulation or legal requirements will be mandatory deployed to all tenants without giving them a chance to refuse (ASAP deployment).

In all deployment cases, a suitable timeframe for parallel run needs to be incorporated to test with the system’s general stability and compatibility with tenant’s own business processes.

## Concern Mitigation Strategies

Each tenant will have content elements configured on individual basis. Sub-sites of tenants will inherit their design and can override any customization (except when locked)

Each tenant will have branding elements imported from their own user-guide to match their own corporate site.

FIS can offer license based deployment to their tenants, with each license having different modules and at different levels of customization. Each module will be developed as a plugin and will be deployable as a standalone service component to the application. Only subscribing tenants can be allowed usage of that module. With database components tied to a module, upon change in license subscription; appropriate changes can be made to different tiers of the application in tenant’s configuration settings.

In case a tenant requests a specific feature or enhancement of existing feature, which will lead to a new version of a plugin or a new plugin altogether. In such case, the plugin will be loaded at runtime for the request concerning the requesting tenant. The database existing logical separation through schema will facilitate independent changes at database level. The application changes will be implemented using “Decorator pattern” over existing business interfaces. Hence once the changes requested get ready for deployment, the same will be detected and activated by “Dependency Injection”. This way, each tenant can have their own business components running in parallel along with core functionalities.

This tenant specific loading will be maintained till the tenant is live in service with FIS. In case FIS deems it fit for the specific change to be offered to other tenants as well, it can offer them as add-ons by including them in their package.

Each plugin’s functionality will be exposed as a SOA service, with endpoints pointing to specific versions. Tenant’s configuration will drive which version gets consumed (base or specifically requested one). In case a new version is released for core functionality, all tenants will be automatically migrated to new version, and previous version will be archived.

## Research Areas

* Plugin architecture
* Endpoints with appropriate service architecture of choice (WCF REST service / Service Stack / Web API / Sockets etc)
* DI loading of tenant specific module through its endpoint.