ptc university

ThingWorx: Fundamentals of Deployment

Instructor-Led Course

THINGWORX: FUNDAMENTALS OF DEPLOYMENT

COURSE OVERVIEW

Take a ThingWorx application through its entire life cycle, exporting it from development and migrating it to production. Follow the best practices for creating stable, deployable, and maintainable applications in the process.

Prerequisites

- ThingWorx: Fundamentals of Modeling 2

Topics

- Development
- Data Migration
- Quality Assurance/Staging
- Logs and Reports
- Production

ptc university

INTRODUCTIONS

Name

Location

Industry

Role

Use the annotation tools to indicate your level of product knowledge.

Beginner	Intermediate	Advanced	Transfer
Less than 100 hours hands-on in the product.	Able to perform necessary tasks in the product.	Several years regularly using the product.	Intermediate/expert in comparable non-PTC software.

ptc university

COURSE INTRODUCTION

HIGHLIGHTS

This section will introduce the four servers and the application we will discuss and use in the training environment.

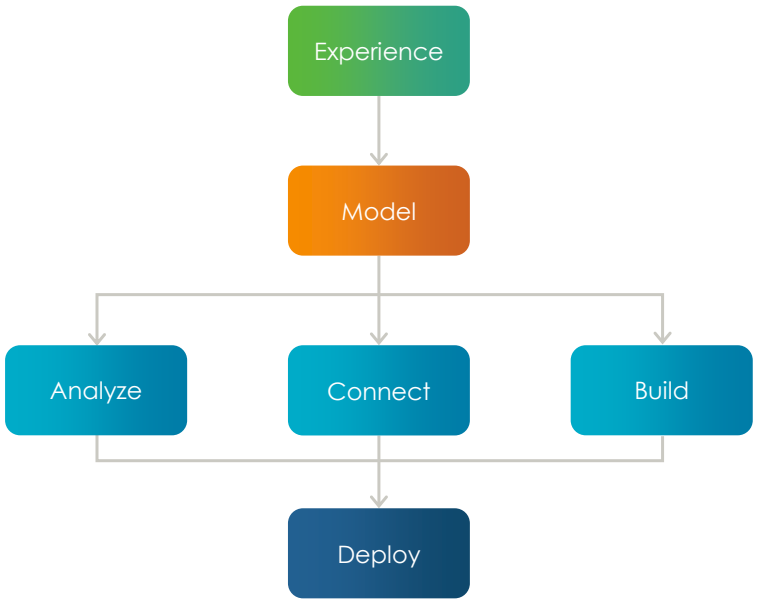
ptc university

COURSE INTRODUCTION

REMINDER: EXPERIENCE BEFORE DEVELOPMENT

Pre-planning is critical to creating the correct functionality for the user. Before starting development, consider:

- Users
- Groups
- Organizations
- Roles
- Problem statements
- Use cases
- Planned functionality
- Branding, styling, and color schemes



ptc university

This section is primarily about development. However, it is still important to follow the ThingWorx Development Process. If you do not do proper planning for your application, you will create the wrong functionality for the user.

Before starting development, you must consider your users – the groups, organizations, and roles they fill. You must consider the problems they need to solve and select appropriate functionality to solve those problems.

Less importantly, many organizations like to standardize on a Mashup color scheme during the experience stage, planning the fonts and styles used in Mashups.

COURSE INTRODUCTION

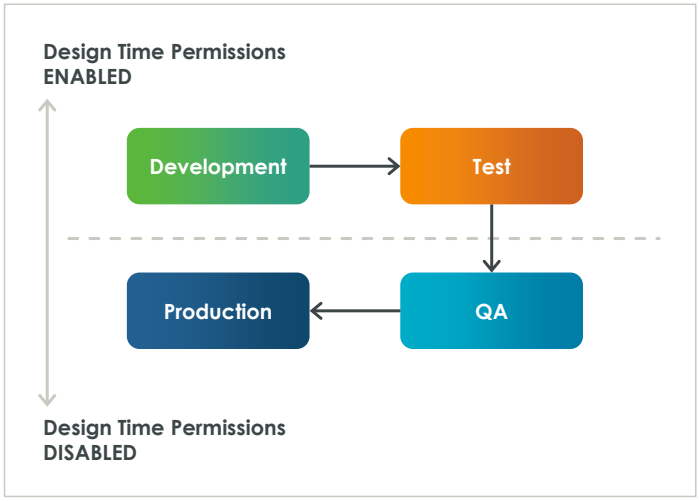
FOUR SERVERS

A minimum of four ThingWorx servers are recommended to release a ThingWorx application to production:

- Development server.
- Data Migration server.
- Formal quality assurance (QA) server.
- Production server.

Design time permissions:

- Enabled in development and test.
- Use the change control system for QA and production changes.



ptc university

As a best practice, a minimum of four ThingWorx servers should be used. This also serves as the main workflow and guide for this course. As we cover each server, we learn the critical tasks that need to be considered during that stage.

These stages are:

- Development – This is where application development happens. This is where you create your Things, Thing Templates, Mashups, and all the aspects of your ThingWorx application.
- Test – This is where features that may affect development, such as new extensions, may be tested. Unit and functionality testing may also occur on the Data Migration server.
- QA – This is where the application is formally tested, including scalability, user acceptance, and performance testing. The QA server also helps formalize the strategy for migrating to the production server.
- Production – This is where the final production application resides.

The development and test servers are informal, so design time permissions should be turned on, enabling developers to make changes to the ThingWorx application.

However, once the application reaches the QA server, design time permissions should be turned off, preventing developers from making changes.

As a best practice, model and application changes should not be made in QA or production. Issues in these stages should be logged in a change control system and resolved on the development or test servers.

COURSE INTRODUCTION

TRAIN ENVIRONMENT

Your training environment enables you to switch between four ThingWorx databases.

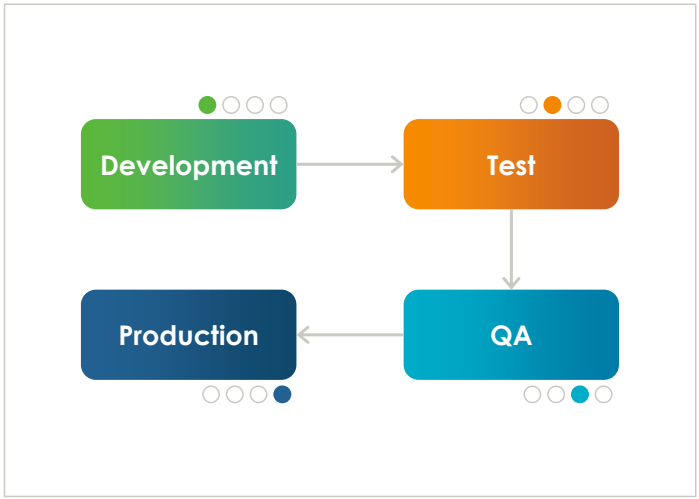
SwitchTWXInstance:

1. Production(Default)
2. Development
3. Test
4. QA

Production database runs by default after Windows restart.

Icons indicate which server is relevant to a topic/exercise.

Switch script is unsupported and should not be used in production.



ptc university

Your training environment enables you to switch ThingWorx databases using an unsupported script to simulate having four servers. Each of the four databases represents a server you would use in a real-world scenario. This mechanism for switching ThingWorx datastores is not supported and should never be used in production. It manipulates the ThingWorx database at the database level, which should only be done when instructed to do so by PTC Customer Support. Manipulating the ThingWorx database at the database level can result in data loss and unpredictable outcomes.

To switch databases, run the SwitchTWXInstance script located on the Windows desktop, and select the database.

The production database is used by default. It will automatically be used if you restart Windows.

It is critical that you complete your exercises in the correct database. It may be difficult for you to tell which database is being used since they all look identical and have the same address (http://localhost/Thingworx). If you are in doubt, re-run the SwitchTWXInstance script.

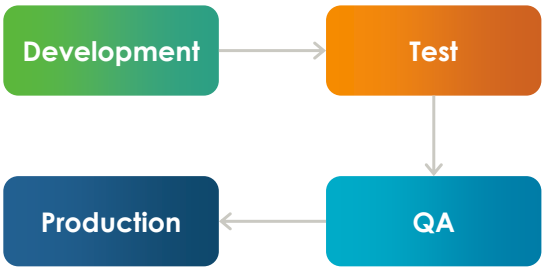
Most slides in this course have an icon indicating which server the topic is discussing or should be used for the exercise.

KNOWLEDGE CHECK

1

On which server(s) should application issues not be corrected but reported to a change control system instead? *Select all that apply.*

A. Development
B. Test
C. QA
D. Production



ptc university

EXERCISE 1 SET UP ENVIRONMENT

1 Start the Development Server

2 Load Lab Files

ptc university

Task 1: Start the Development Server

1. Close all web browsers.
2. Open Windows File Explorer. Then, double-click the **W:\TWFD-DPLY-Lab-Files\SwitchTWXInstance** shortcut.
3. Press **2** to select the development database.
   Note: If you wait more than ten seconds to press 2, the wrong option will be selected automatically. If that happens, press CTRL+C to abort the script and try again.
5. Start **Google Chrome** and navigate to **chrome://settings/downloads**.
6. Turn on the **Ask where to save each file before downloading** setting.
7. Log into ThingWorx.
   - Username: **Administrator**
   - Password: **ptc-university-123**

   Note: The URL is //localhost:8181/Thingworx/Composer.

Task 2: Load Lab Files

1. As an extension, import **W:\TWFD-DPLY-Lab-Files\1229-CSVParser_Extensions.zip**.
2. As entities, import **W:\TWFD-DPLY-Lab-Files\TWDP-DPLY-Entities.xml**.
3. Navigate to the **Services** page of the **PTC.CS.ColdStorage.Helper** Thing.
4. Copy the 20200616152804 from the lab files folder.
5. Open Windows Explorer and navigate to **C:\Program Files (x86)\ThingWorxFoundation\ThingworxStorage\repository\SystemRepository**
6. Paste the folder here.
7. Execute the **SetupCollectionAndSystemPermissions** service. The service does not return a result.

Additional Information

PTC University uses Google Chrome for ThingWorx training. You may set Chrome as the default browser using the following procedure:

1. On the Windows Desktop, select Start > Settings.
2. Select Apps.
3. Select Default apps.
4. Click Internet Explorer.

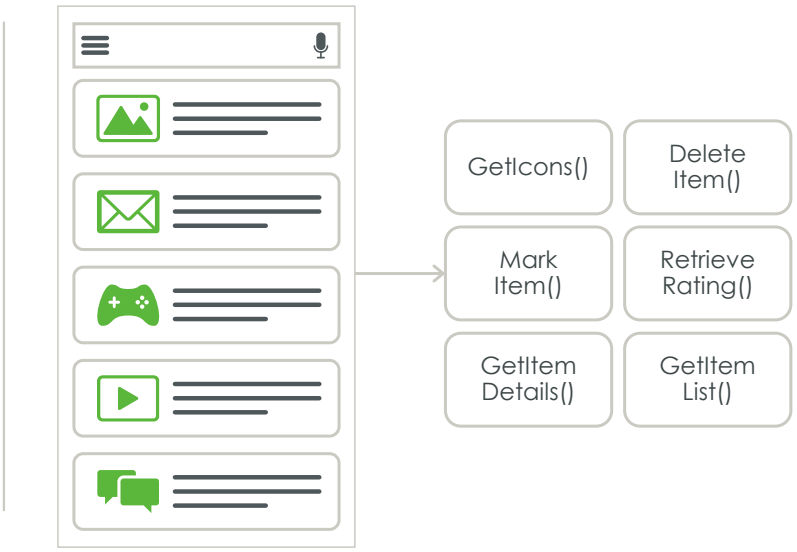
5. Select Google Chrome.
6. Close the Settings window.

ThingWorx fully supports Firefox, Google Chrome, Microsoft Edge, and Safari browsers.

COURSE INTRODUCTION

BEST PRACTICE: MASHUP- BASED DEVELOPMENT

If it is a user-interface-focused use case, focus on developing functionality for the UI. This will avoid wasting time collecting data and modeling features that the user interface or the application will not use.



ptc university

During the experience stage, a proven practice is identifying the features required in the user interface and staying focused on developing those features throughout development.

This will avoid wasting time collecting data and modeling features that the user interface or the application will not use.

However, be aware that there are times to violate this because some use cases are not UI-focused. If your application focuses highly on automation without user interaction or big data analysis, Mashup-focused development may not be appropriate.

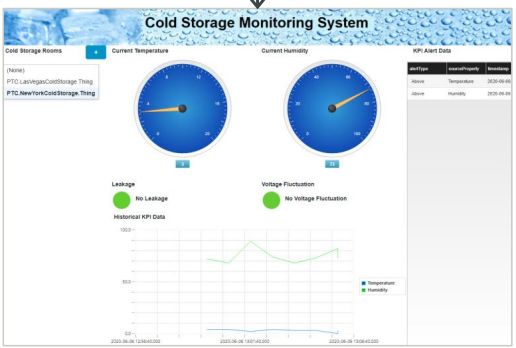
PTC COLD STORAGE

Exercise Scenario

PTC Cold Storage has storage rooms in New York, Los Angeles, and Las Vegas.

Recently developed IoT system monitors cooling failures, voltage fluctuation, temperature, humidity, and leakage to maintain quality and prevent inventory loss.

The application needs to be tested and put into production.



ptc university

In this exercise, we migrate the PTC Cold Storage application.

This organization runs several cold storage facilities in various states. Each facility must be kept at a specific atmospheric condition and has a sensor for voltage, humidity, refrigerant leakage, and temperature.

The development team has built an initial IoT system to manage this. Right now, it is in early development, only having monitoring capabilities. Still, management has decided that these capabilities can help maintain quality and reduce failure, so they want to put it into production as soon as possible and add more features to the system later.

COURSE INTRODUCTION

PTC COLD STORAGE MODEL USER APPLICATION

Two Mashups:

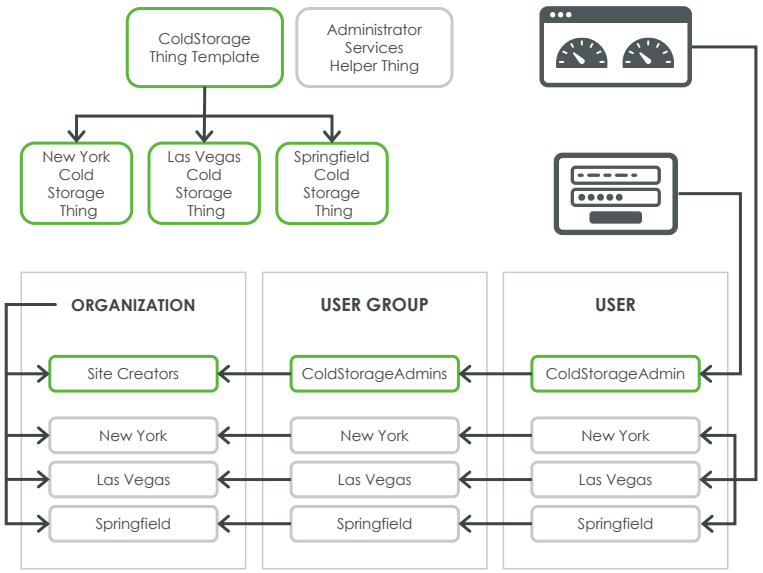
- Monitor site
- Create site

Access Control Entities:

- Cold Storage Admins
- Site Operators

Model Hierarchy:

- Cold storage Thing Template has four properties.
- Site Things derived from template.
- Helper Thing has services to handle setup, Thing provisioning, and site creation.

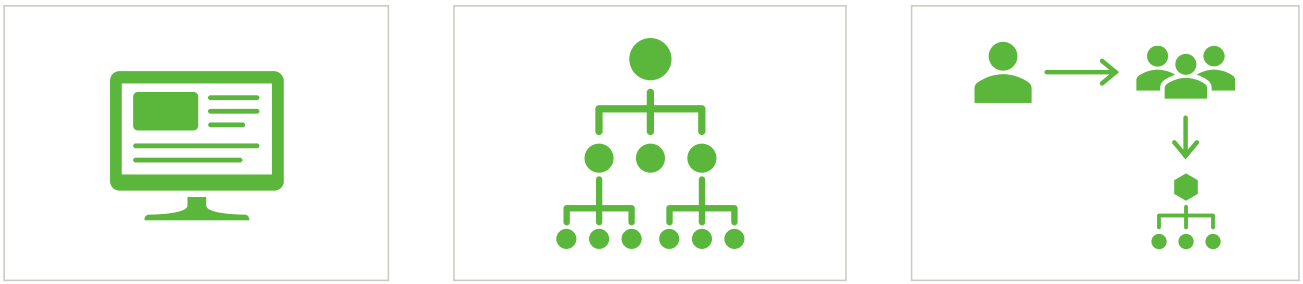


ptc university

In the next exercise, we will examine the cold storage application.

- Mashups: There are two Mashups in this application. One is the main Mashup that enables an operator to select and monitor a site. The second is a dialog window that enables an administrator to create a new site.
- Access Control Entities: There is a cold storage administrator user who can access the Mashup and create new sites. Some site-specific users only have monitor capabilities. These users have corresponding groups and organizational units.
- Model Hierarchy: The model is a Thing Template based upon the parent-child modeling pattern. There is a Thing Template that contains properties for all sites. There is also a Helper Thing that has the provisioning services required to create new sites.

EXERCISE 2 EXPLORE THE PTC COLD STORAGE APPLICATION



1 Use the Mashup to Create a Site

2 Examine the Model Hierarchy

3 Examine Access Control Entities

ptc university

Note: Use the development database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Use the Mashup to Create a Site

1. Navigate to the PTC.CS.ColdStorage.MU design tab and click View Mashup
2. Click the +icon in the upper-left corner next to Cold Storage Rooms.
3. Create a cold storage room with the following attributes:
   - Cold Storage Room Name: **NewYork**
   - Operator Password: **ptc-university-123**
4. Click Submit and dismiss the message indicating the site was created successfully.
5. Close the dialog window.
6. Wait at least one minute for the device simulator to populate the ColdStorage-NewYorkThing with data.

Task 2: Examine the Model Hierarchy

1. Navigate to the **View Relationships** page of the **PTC.CS.ColdStorage.TT** Thing Template.
2. Select **Uses This Entity**.
3. Notice that the only Thing using this template is the **ColdStorage-NewYork** Thing that you just created.
4. Navigate to the **Subscriptions** page of the **PTC.CS.Simulator.Timer** Thing.
5. Select the **SetColdStorageProperties** subscription.
   Note: This subscription is a simulator, setting properties for humidity, temperature, leakage, and voltage during development before a REST API agent is connected. This timer, like most development simulators, would be disabled in production.

Task 3: Examine the Access Control Entities

1. Navigate to the **Organization** page of the **PTC.CS.ColdStorage.Org** organization.
2. Examine the organizational structure.
   Note: The NewYorkColdStorage organizational unit was created when you created the site.
3. Select the **SiteCreators** organizational unit.
4. Click the **PTC.CS.ColdStorageAdministrators.Group** and select **View**.
5. Click **Members** and notice that the only member is the ColdStorageAdministrator user. This is the user that you will use when testing Mashups.

SECTION OVERVIEW

1 DEVELOPMENT

2 DATA MIGRATION

3 QUALITY ASSURANCE/STAGING

4 LOGS AND REPORTS

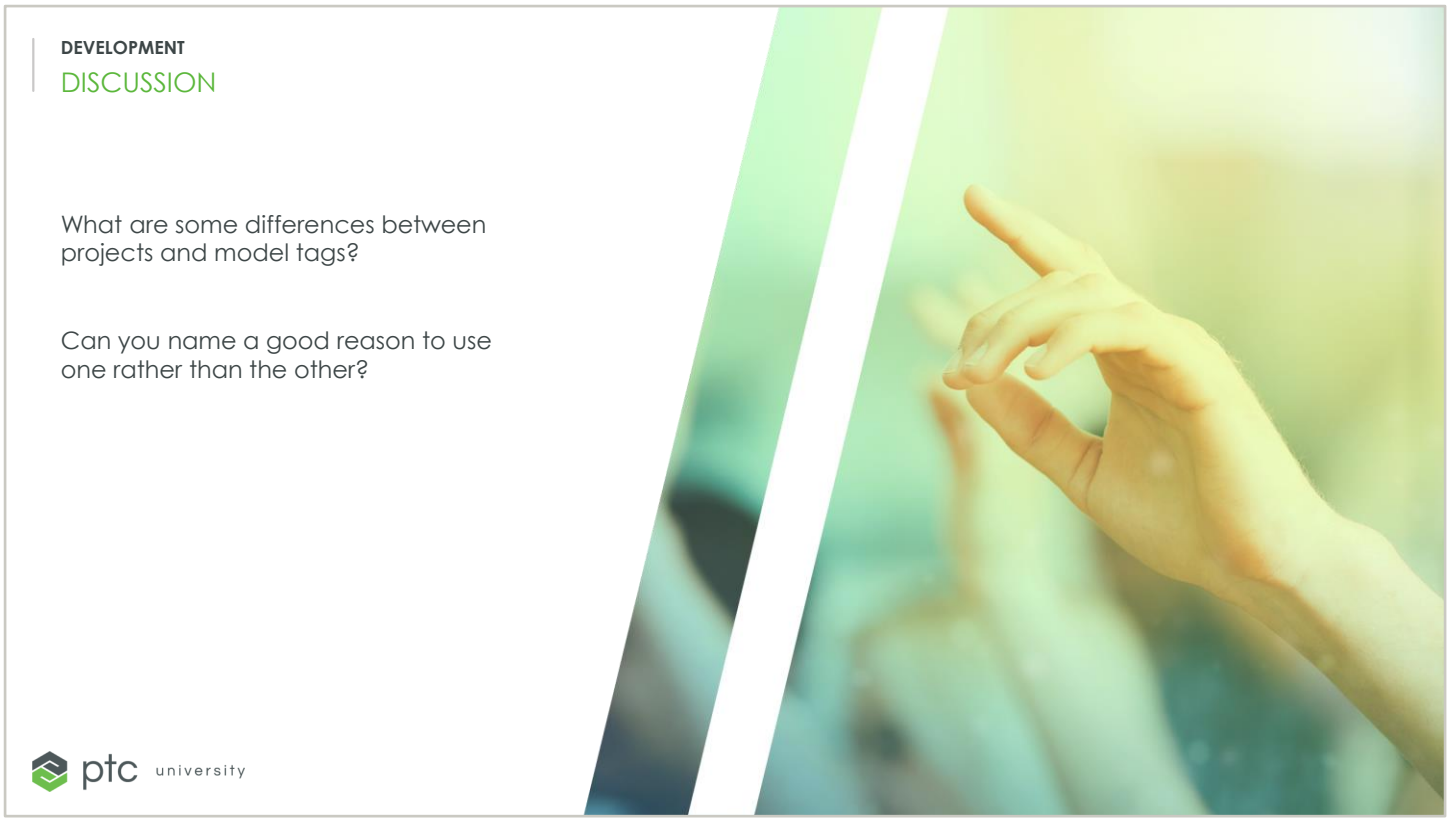
5 PRODUCTION

DEVELOPMENT

HIGHLIGHTS

This section is primarily focused on the development stage of the ThingWorx application. We will look at best practices around development and perform a hands-on exercise where we fix a standards issue with a service.

ptc university



A few differences between projects and model tags are:

- An entity can only be a member of one project but may have many tags.
- You can add/remove project membership from the project information page or use project services. Tags may only be added or removed from the entity's details page or services.
- The project context in Composer enables you to add new entities to a project by default. Tags do not have a similar feature.

Generally, projects are best for bundling up all the entities in your application, while tags are best suited for smaller tasks, categorizing some entities within your application.

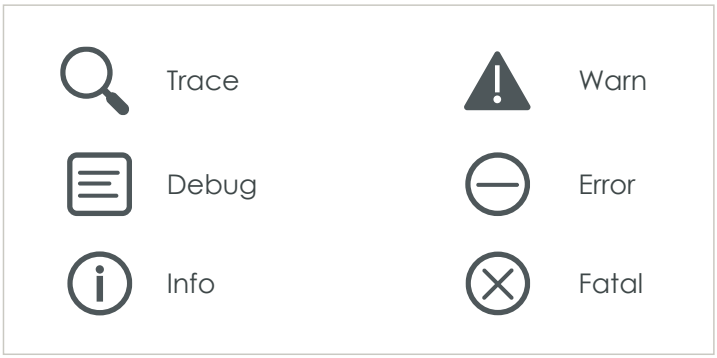
DEVELOPMENT

LOG PRACTICES

Log at ERROR and WARN if the information is useful to a production administrator only.

Log at INFO for information that administrators may find useful when troubleshooting.

Log at DEBUG or TRACE for development troubleshooting.




```
logger.error(userName + "'s attempt to create " +
    thingName + " failed.");

logger.info(userName + " created " +
    thingName + " successfully.");

logger.debug("row " + i +
    " of toolInfoTable populated. ");
```

ptc university

You will use logs frequently when writing services and subscriptions. They are very useful to help troubleshoot your code.

However, it is important to remember that active log commands have a performance drain on the ThingWorx system. Having too many active log commands slows down the system and makes the log more difficult to read.

Most production systems log at the WARN level, so logger messages below are ignored and do not consume system resources. Because of this, you should:

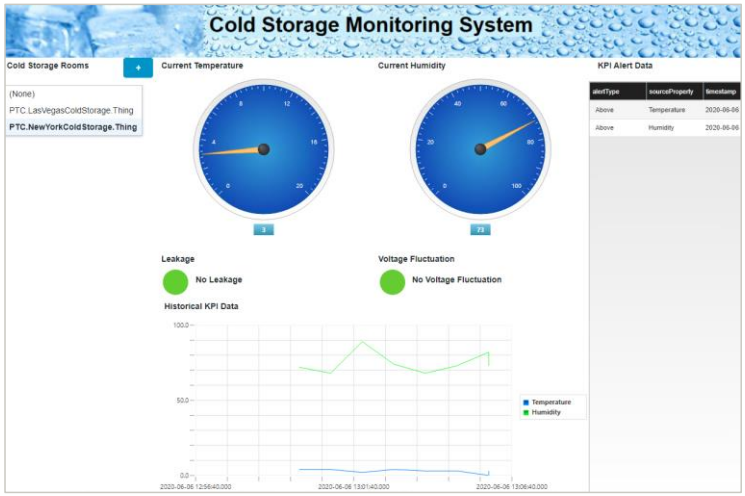
- Avoid logging at WARN or higher unless that information is useful to the ThingWorx administrator.
- Log at INFO if the message might be useful to a ThingWorx administrator doing troubleshooting. Administrators may lower the log level to INFO or DEBUG when they encounter an issue.
- Log at DEBUG or TRACE during development for monitoring that your service is working as expected, such as tracing variables or tracking the execution flow of a loop.

DEVELOPMENT

BEST PRACTICES: DEVELOPERS TEST AS NON-ADMINISTRATIVE USER

One critical best practice is that every developer should test their code on the development server. When testing their code, developers should:

- perform basic functionality tests on the development server.
- test Services as a non-administrative user.
- create a user in the same manner as your application's administrator would.



Do not use the View Mashup button.
Test services and Mashup as a non-administrative user.

ptc university

One critical best practice that many developers ignore is every developer tests their code on the development server. It's extraordinarily difficult to write code successfully without running it, and since we've advanced past developing software using punch cards, developers do not have to.

However, as a developer, do not assume that if your service or Mashup works for the administrator, it will work for your users. ThingWorx access control is very powerful but also very precise. When you test as an administrator, you fail to test your access control procedures.

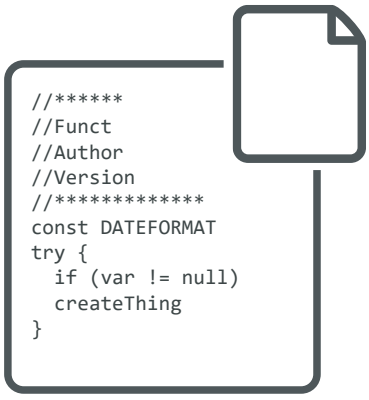
DEVELOPMENT

BEST PRACTICE: CODE REVIEW HELP ENFORCE STANDARDS

Including the best practices previously mentioned, the one with the most impact is to conduct live code reviews.

Code Reviews:

- Help developers maintain standards.
- Use peer pressure to help improve the quality and efficiency of code.
- All codes should be reviewed by several team members.



ptc university

Of these best practices, the single one with the most impact is to conduct live code reviews.

A couple of other developers should review every developer's code. Code reviews offer several benefits in addition to helping enforce the other standards:

- It is proven that the additional input results in fewer errors and more efficient code.
- It helps developers learn from each other.
- Up to 75% of the issues found during code review are not functionality issues but evolvability/maintainability issues. If your application needs to evolve, code reviews simplify that.

SECTION OVERVIEW

1 DEVELOPMENT

2 DATA MIGRATION

3 QUALITY ASSURANCE/STAGING

4 LOGS AND REPORTS

5 PRODUCTION

DATA MIGRATION

HIGHLIGHTS

In this section, we will dive into application import and export options, best practices, and application testing.

ptc university

DATA MIGRATION

HOW SCMS AND THINGWORX INTERACT - COMPONENTS

Persistence Provider

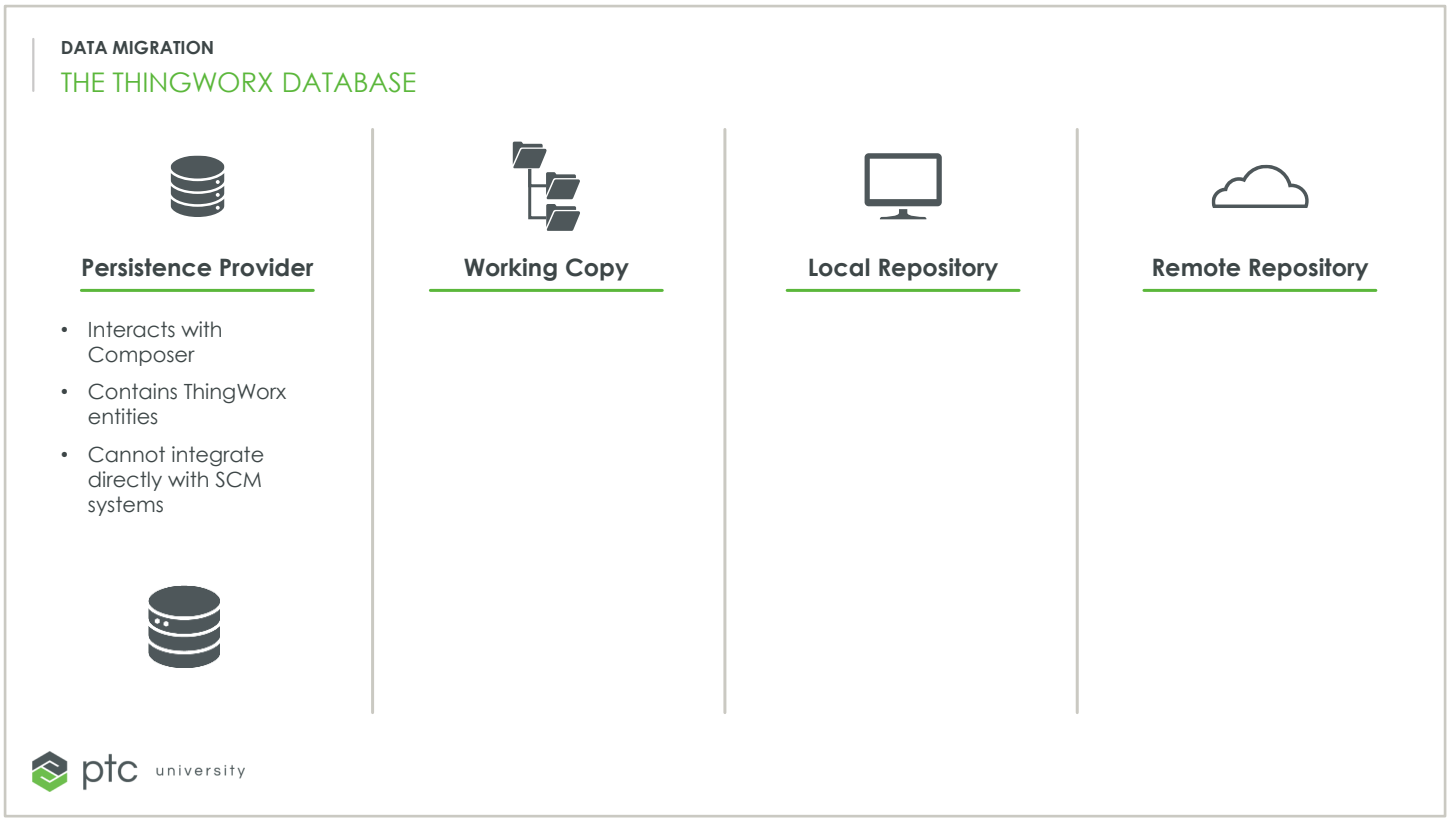
Working Copy

Local Repository

Remote Repository

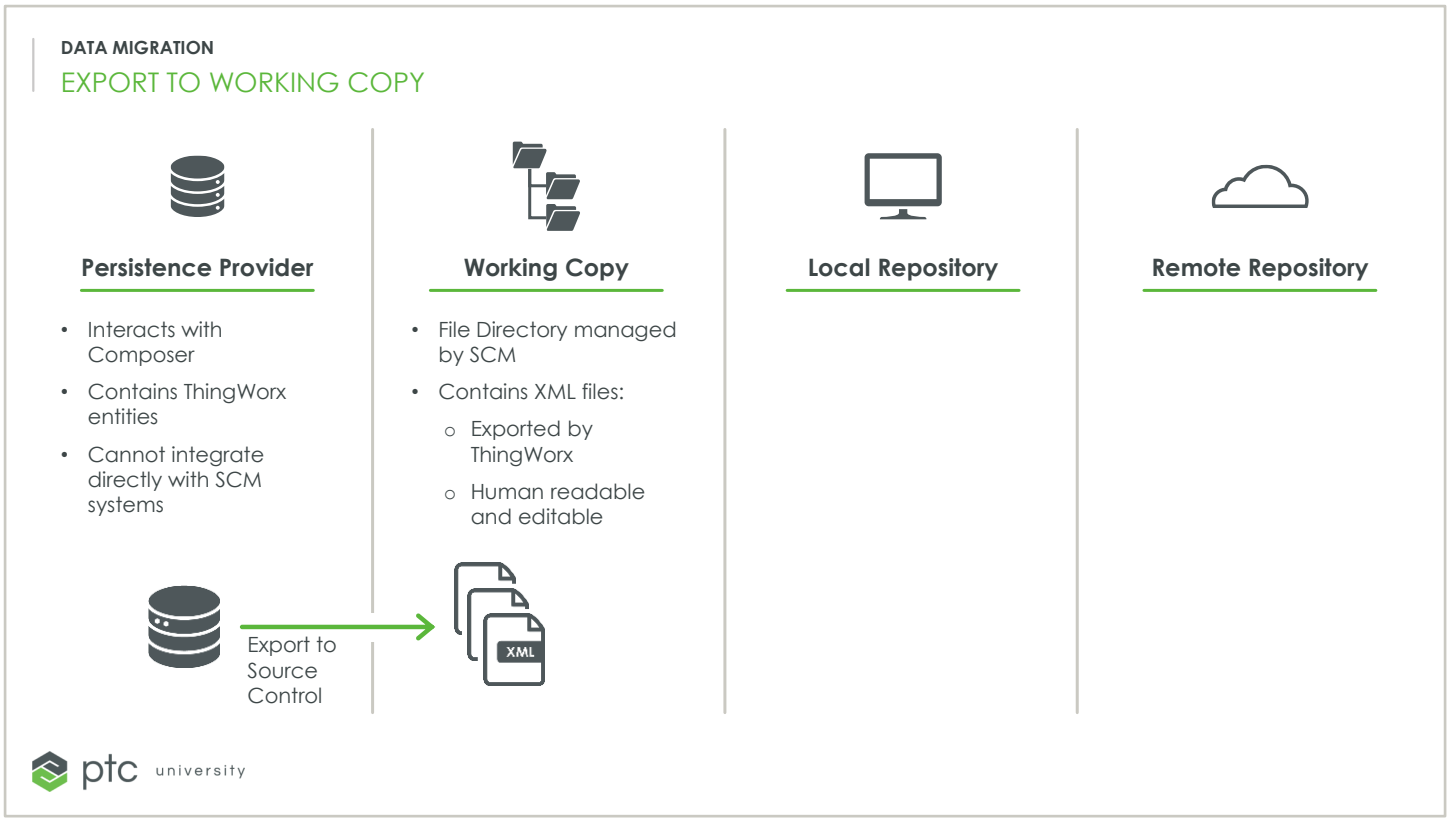
ptc university

When working with a SCM, your data is in four places at once, each in a slightly different form and state.



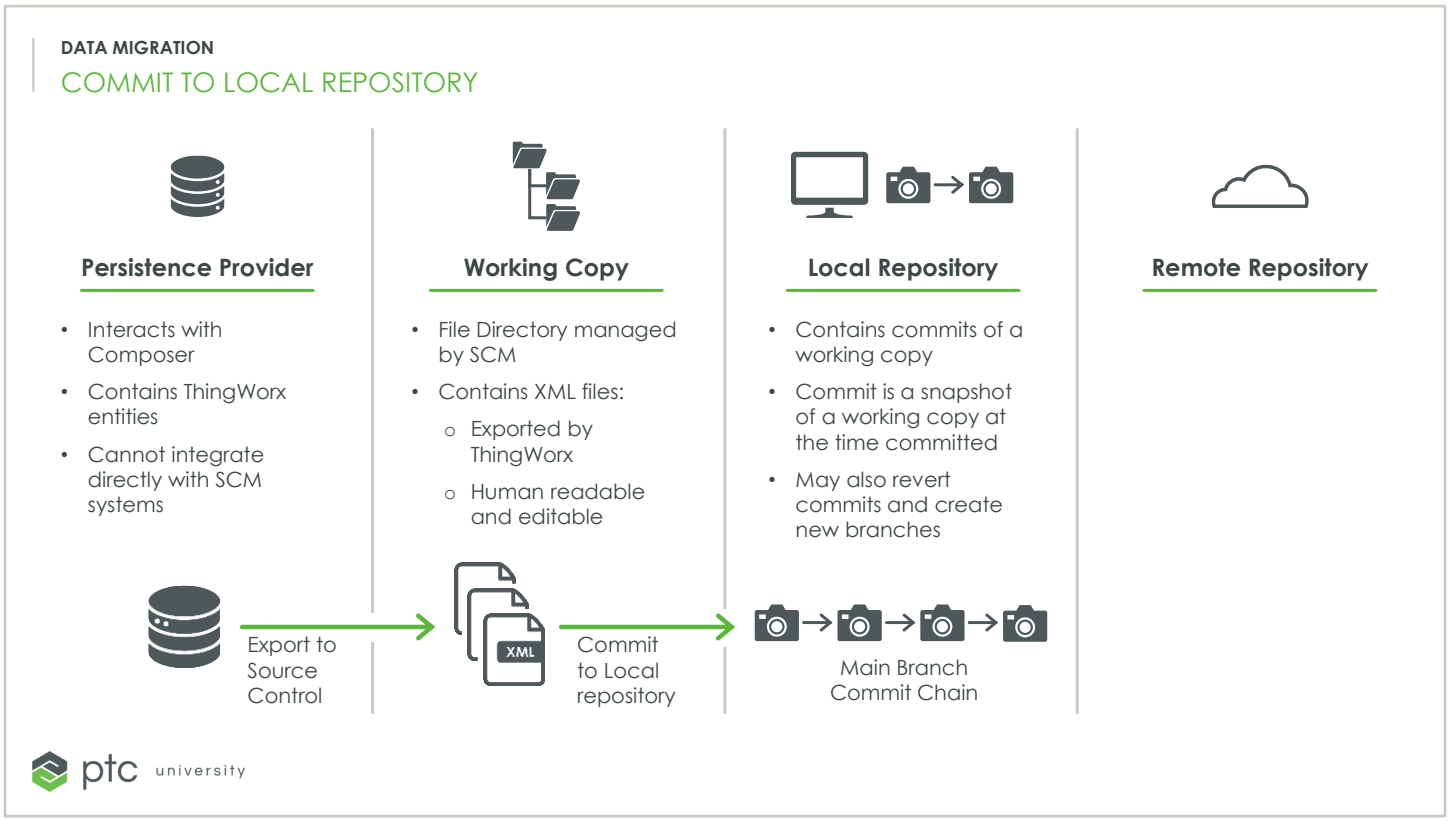
The persistence provider is the database ThingWorx uses to store its entities. When you modify an entity in Composer, you are modifying this database.

However, SCM systems cannot directly manage ThingWorx entities in the database.



To get persistence provider information in a format that a SCM system can work with, you must export your entities to source control.

This saves each entity as an independent XML file. The SCM system can manage these XML files. They are also human-readable and may be edited in any text editor.



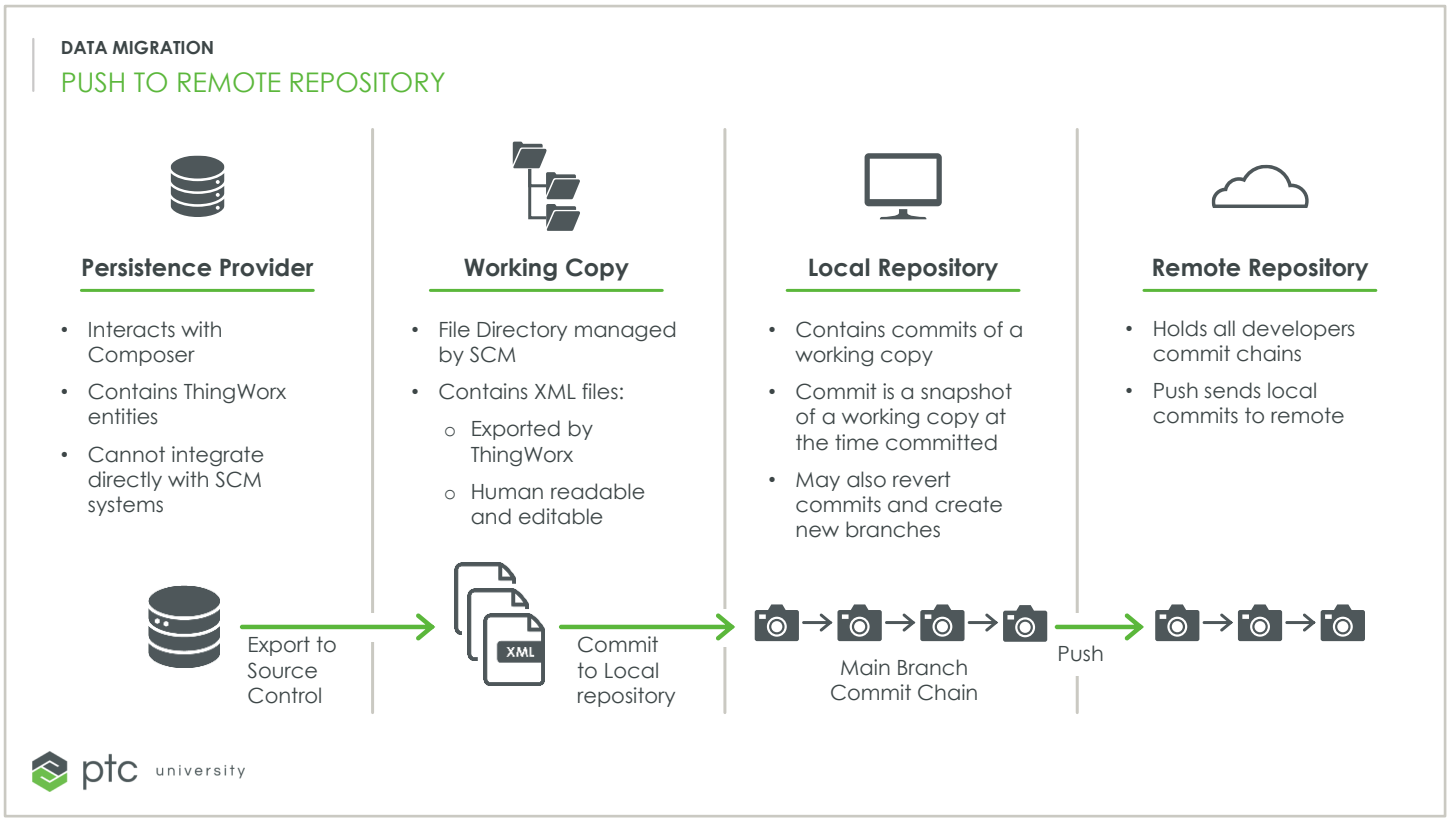
The local repository is where you start to get SCM functionality. This is an instance of a SCM installed on the developer's machine.

Using a SCM to commit your working copy creates a commit point in the local repository. A commit point is a snapshot of the working copy at the time the commit was taken. It records who performed the commit and all changes to the files.

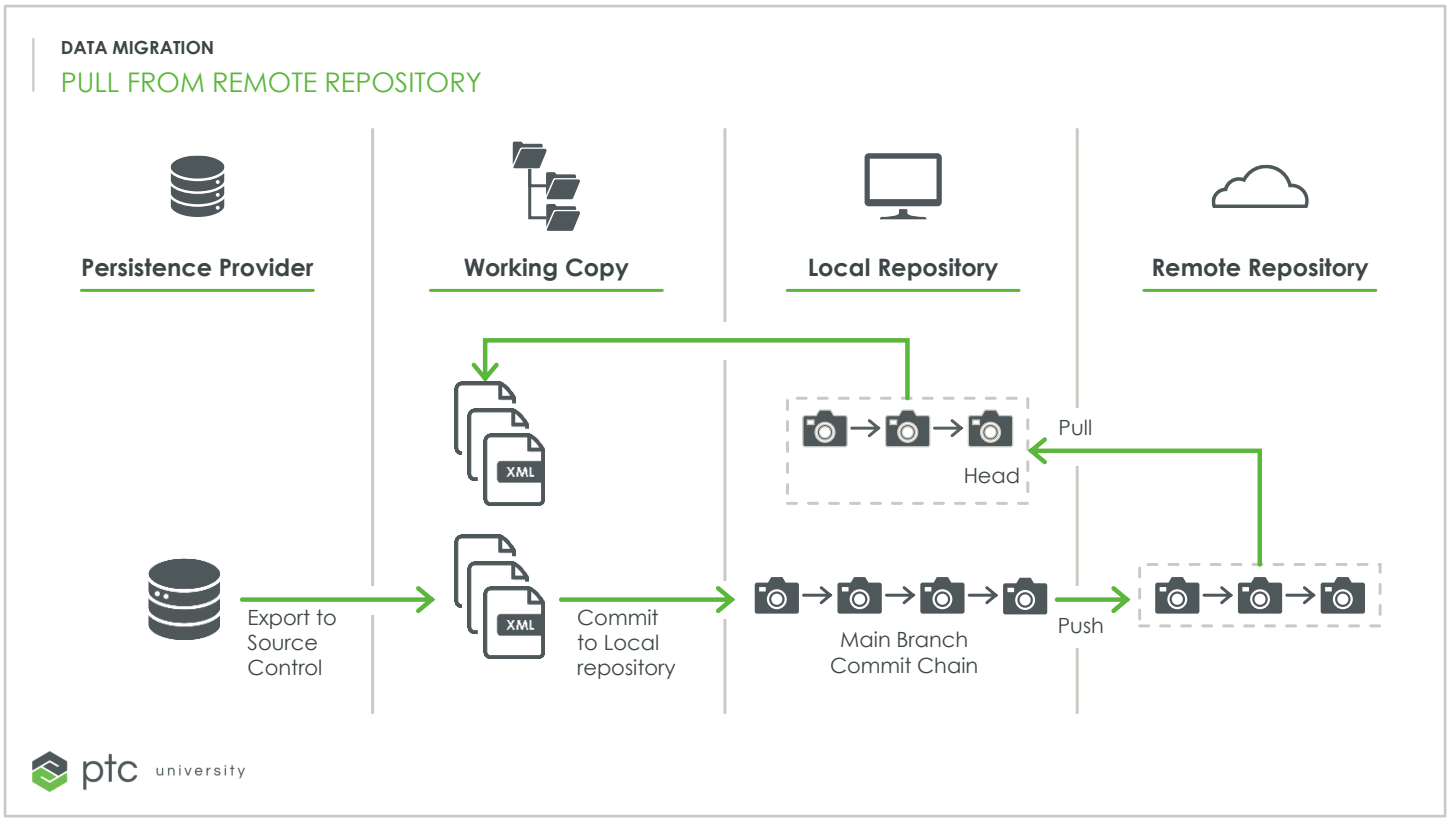
This enables you to examine the commit chain, determine who made every change, and when that change was made. You can also revert to a previous commit, placing the code as it was at the time of that commit into your working copy.

You may also branch your repository. We will not be branching in this course, but branching creates a separate commit chain where you can independently work on functionality. The branch may be merged back into the main branch at a later point.

One critical architectural difference between a SCM and Subversion is that Subversion does not have a local repository. In Subversion, all commits are made directly to the remote repository. This enables you to work with a SCM offline and provides superior performance when committing large projects. However, Subversion SCM systems cannot be out-of-sync and are simpler to use.



The remote repository stores commits from all developers. A push in a SCM sends your local commits to the remote repository.



After commits have been pushed to remote, they may be pulled by other developers.

A pull takes the new commits from the remote repository and synchronizes them with the local repository. Then, the most recent snapshot, called the head, is synchronized with the working copy.

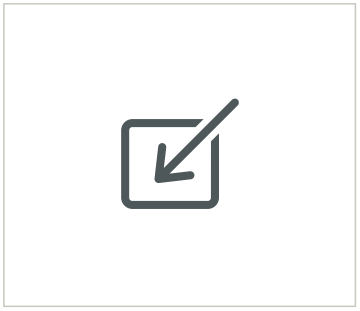
At this point, the new developer has a set of import files for ThingWorx.

DATA MIGRATION

EXPORT OPTIONS

There are several options when exporting a ThingWorx application. Those options include:

- All entities
- Project
- Individual entities:
  - Source code control
- Solution Package/Solution Central



Import

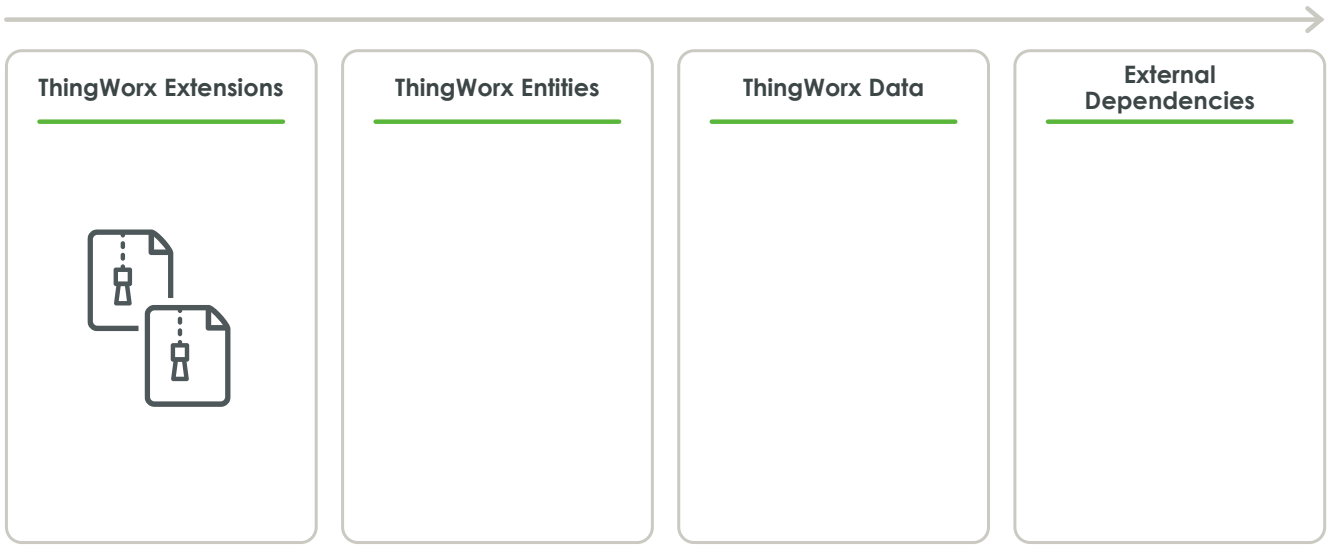
Export

ptc university

There are several ways to export your application from the development server. Each has distinct advantages and disadvantages.

DATA MIGRATION

EXPORT A THINGWORX APPLICATION



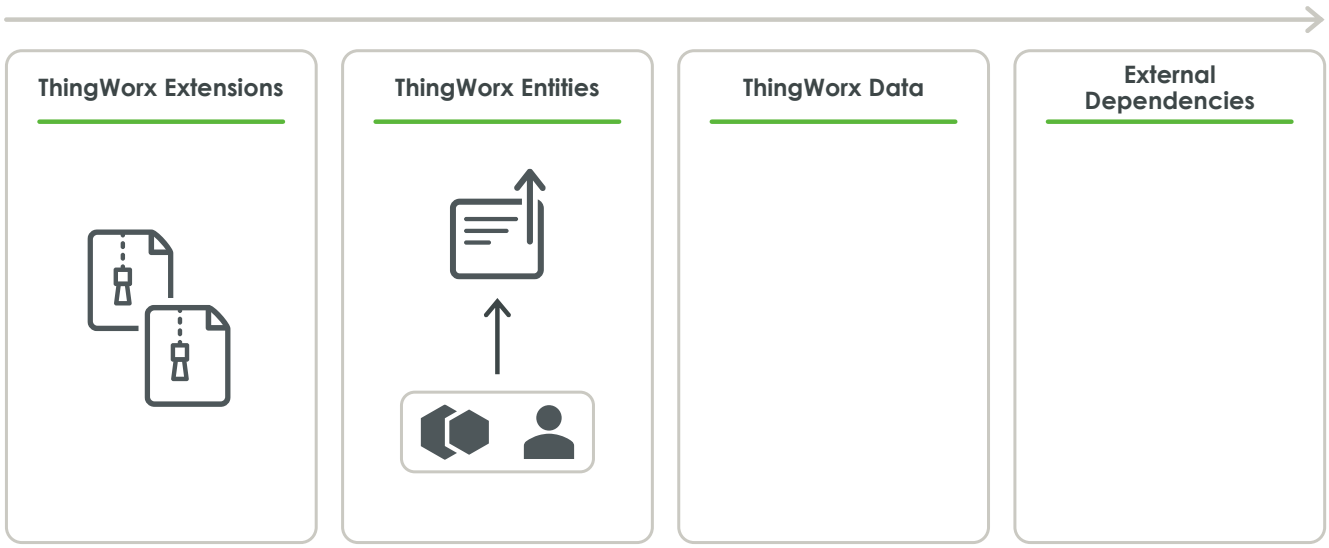
ptc university

If you took the Fundamentals of Connectivity course, you learned how to manage extensions. These are ZIP files containing code that adds extra functionality to ThingWorx, such as sending an email, downloading files from third-party solutions, and many other add-on functions.

Any system that uses your application will need the same extensions set, so keep them stored for later use. These extensions cannot be placed in a ThingWorx project.

DATA MIGRATION

EXPORT A THINGWORX APPLICATION



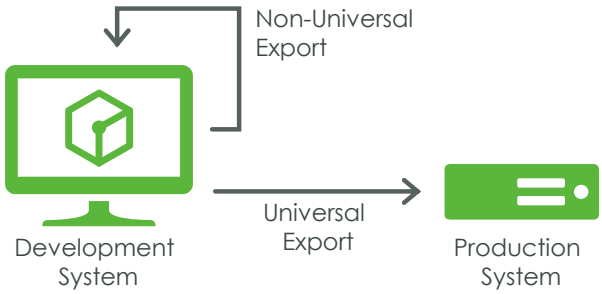
ptc university

Next, you need to export the entities for your ThingWorx application. This is where the project you created earlier will come in handy. Export all the entities, such as Things, users, and Thing Templates, in the project, and you have a neat package ready for import into a new system.

DATA MIGRATION

UNIVERSAL ENTITY EXPORT

When exporting entities, one important option is universal export. Exports without Universal Export are more secure but cannot be used to rehost an application. If you plan on rehosting, you must use universal export but be aware that the files contain sensitive data.



ptc university

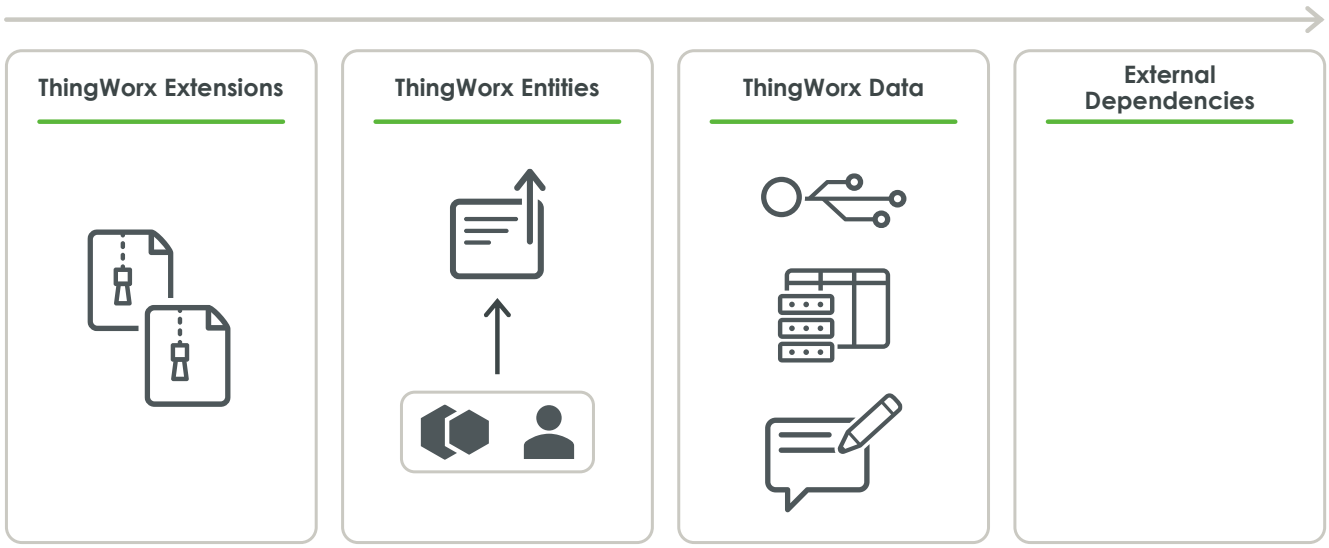
When exporting entities, one important option is universal export.

Non-universal export fully encrypts sensitive data such as user passwords. This makes it far more secure than universal export, and for that reason, non-administrative uses can export entities this way. However, these exports are only suitable for creating backups that are imported back into the original system. They may not be used to rehost entities to another system, such as moving entities from development to production. This is because the target system won't have the keys necessary to decrypt the sensitive data.

So, if you plan on rehosting, you must use universal export. Be aware that the export files contain sensitive information, so treat them appropriately. You need to secure the export files and not put them in web-accessible locations.

DATA MIGRATION

EXPORT A THINGWORX APPLICATION



ptc university

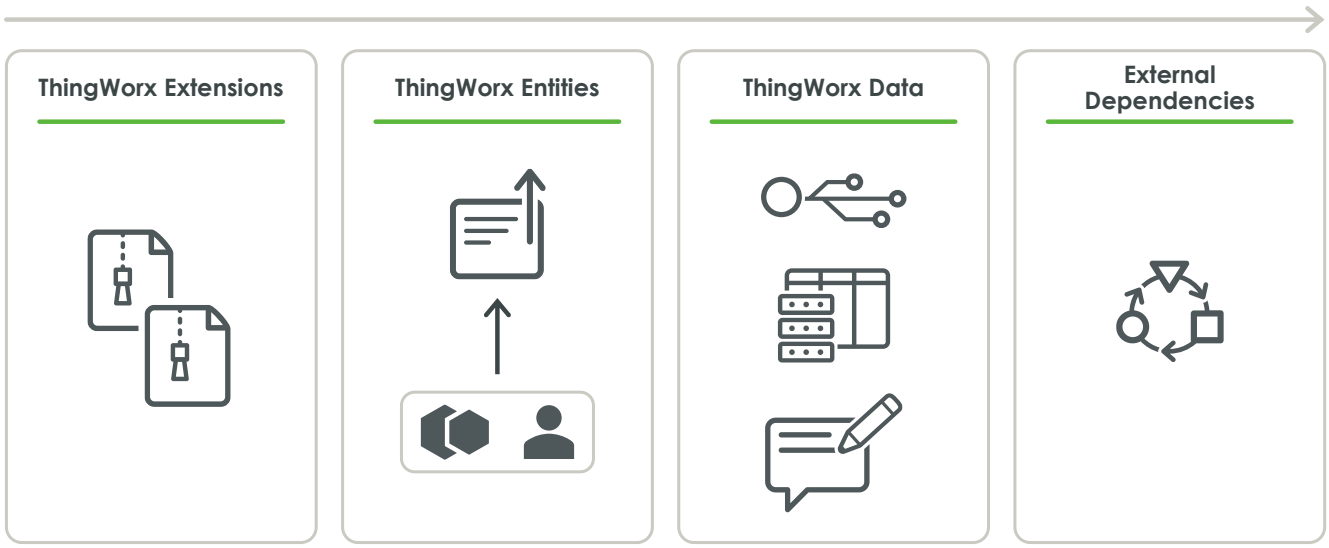
Some entities are used to manage large amounts of data. Things and Thing Templates do not, but a few data storage entities, such as Streams, Data Tables, and Blogs, contain data.

When you export a data entity such as a Stream, you export the entity's definition but not the data inside it. This is an important feature, as these entities can contain a great deal of data that would take a long time to import and export. When you move a system, you often want to migrate the application, but not the data.

If you do want to migrate the data, you must export the data separately.

DATA MIGRATION

EXPORT A THINGWORX APPLICATION



ptc university

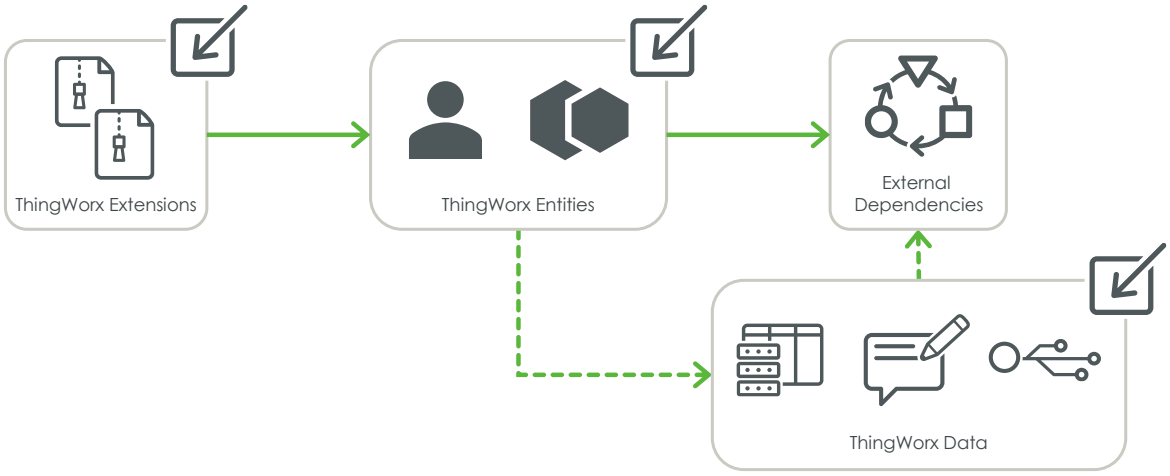
ThingWorx is an application development tool, and many developers add dependencies to their applications.

For example, they may install an optimized database for a specific type of data and write services that access that database.

If you have these external dependencies, you must consider them in your import/export strategy.

DATA MIGRATION

IMPORT A THINGWORX APPLICATION



ptc university

Next, we will discuss importing your application.

When importing your application, you must do so in a specific order.

First, import your extensions. Entities won't import if the extensions they rely on are not already in the system.

Second, import your entities.

Third, import your data, populating those entities. You can skip this if you do not want data from the old system in the new one.

And lastly, verify any external dependencies that you built into your code are available.

DATA MIGRATION

IMPORT OVERWRITES EXISTING ENTITIES

Import overwrites existing entities with the same entity name:

- No warning
- Entity completely replaced by imported entity
- No merge functionality

Items in entity:

- Identifiers
- Properties/Services
- Events/Subscriptions
- Configuration
- Permissions

Warning: Possible Loss of Data

ptc university

One crucial warning when doing import and export.

If the import file contains an entity with the same name as one that already exists in the system, it will be overwritten. The imported entity will completely replace the pre-existing entity. The two entities will not be merged.

There is no warning that the entity is going to be replaced.

This is true even with system entities, such as subsystems, so care should be taken not to adversely affect a target system, such as importing development queue performance settings into a production system designed to handle much more traffic.

Lastly, as a quick review, an entity includes not only properties, services, events, and subscriptions, but also:

- Access control permissions, such as visibility and run time permissions.
- Identification information on the general information page, such as the name, identifier, and value stream used.
- Configuration tables, which change depending upon the entity type. This could be connection information for a database Thing; a password for a user.

DATA MIGRATION

EXPORT ALL ENTITIES TO FILE

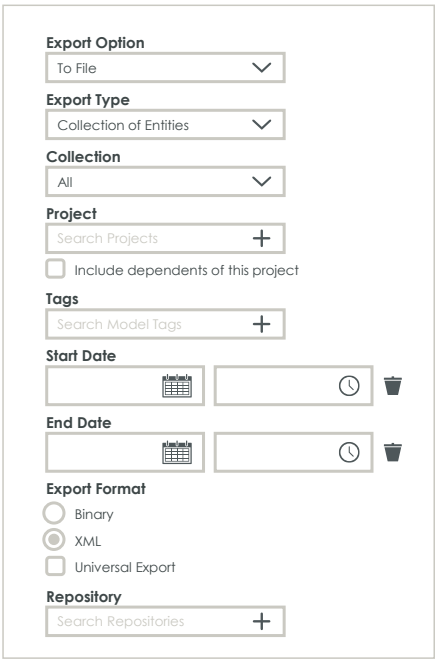
When exporting all entities, some entities will not be included, and some entities that may cause unwanted effects on the target system.

Not Included:

- Extensions
- Data
- File Repository Contents
- Collection Permissions

Included:

- System entities
  - Subsystems
  - SYSTEM user
  - Administrator, including password



ptc university

Exporting all entities cannot be used to replicate your development ThingWorx system on any other system perfectly. Some entities will not be included, and other entities may cause unwanted effects on the target system.

Some important items that are not exported when exporting all entities are:

- Extensions are not included in the export. If your entities depend upon extensions, they must be installed before import.
- Data inside blogs, wikis, streams, and data tables is not included. Data must be exported separately.
- File repository entities are exported, but the files inside the repositories are not. These files must be copied from the source to the target server.
- Collections are not entities and are not exported. So, if you set collection permissions in the source system, you must replicate those permissions on the target system. Collection permissions also affect all ThingWorx applications on the server. If your server hosts more than one application, changing collection permissions may adversely affect other applications.

An export of all entities may also include items that you do not want imported into the target system.

- Subsystem permissions and settings are exported. These are instance-wide settings, so changes to them may affect other applications on the target server.
- Permissions given to the SYSTEM user are exported. This is usually good unless another application on the server has intentionally denied permissions to the SYSTEM user.
- The administrator user is exported, including the administrator password. Importing into the target system will change the administrator password.

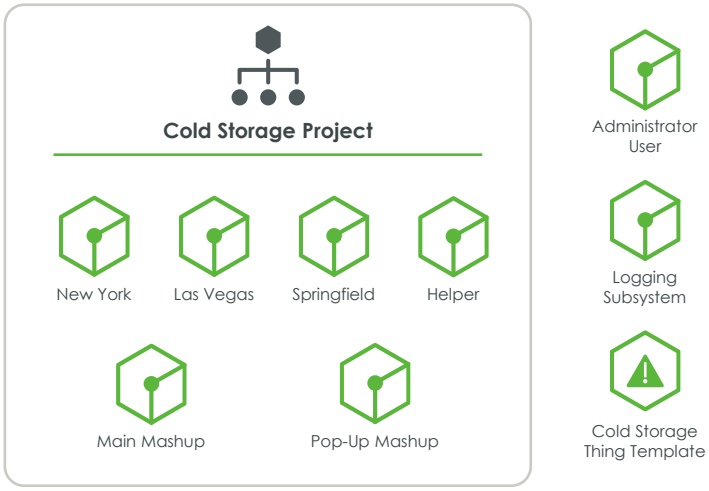
DATA MIGRATION

BEST PRACTICE: EXPORT PROJECT

Exporting the project is the preferred method for migrating a single development system to test. Examples of system entities that can not be placed in a project include:

- Extensions
- Data
- File Repository Contents
- Collection Permissions
- System Entities

If you do not put dependencies in the project, it may result in incomplete export. This means that the export will work, but the import may fail due to missing dependencies.



ptc university

Exporting the project is usually a better practice than exporting all entities.

The primary advantage is that system entities cannot be tagged or placed in a project. As a result, you can be assured that the export file will not overwrite any critical items, such as subsystem settings or the administrator password. In addition, the export file is smaller and imports faster.

However, there is one potential disadvantage. If you do not tag dependencies or put them in a project, you may get an incomplete export that cannot be imported.

Let us examine the example in the slide. Most of the entities needed for the project are in the project.

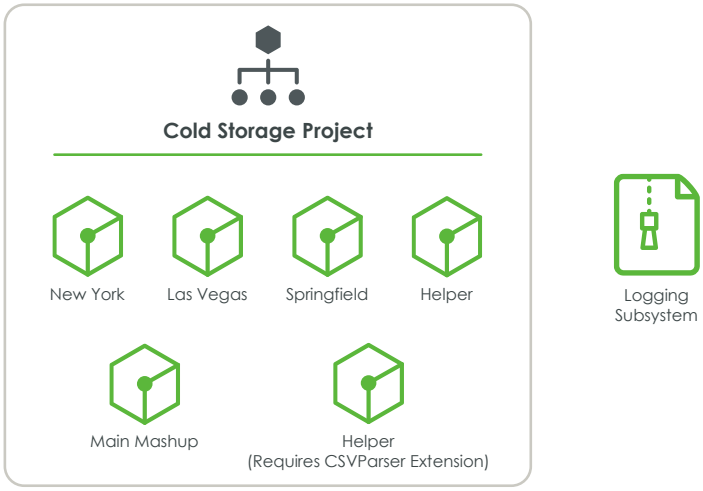
However, some entities are not:
- System entities, such as the Administrator user and Logging subsystem, cannot be placed in a project. That is usually okay because every target ThingWorx system will have these default entities.
- The cold storage Thing Template has mistakenly not been included in the project.

When you import one of the site Things that use the cold storage Thing Template, the import will fail. This is because ThingWorx cannot create the site Thing without its parent Thing Template. Similar failures would occur if permissions referred to a group or organization that did not exist.

DATA MIGRATION

EXTENSIONS

Extensions cannot be included in entity exports, but some entities require them to exist to be imported. This means you should always import extensions into a new system first.



ptc university

Extensions cannot be included in entity exports, but some entities require them to exist to be imported.

As an example, our cold storage helper entity has a service that uses the CSVParser extension. If that extension isn't in the ThingWorx system, the helper entity will fail to import.

The solution to this is always to import extensions into a new system first, assuring they exist and the import succeeds.

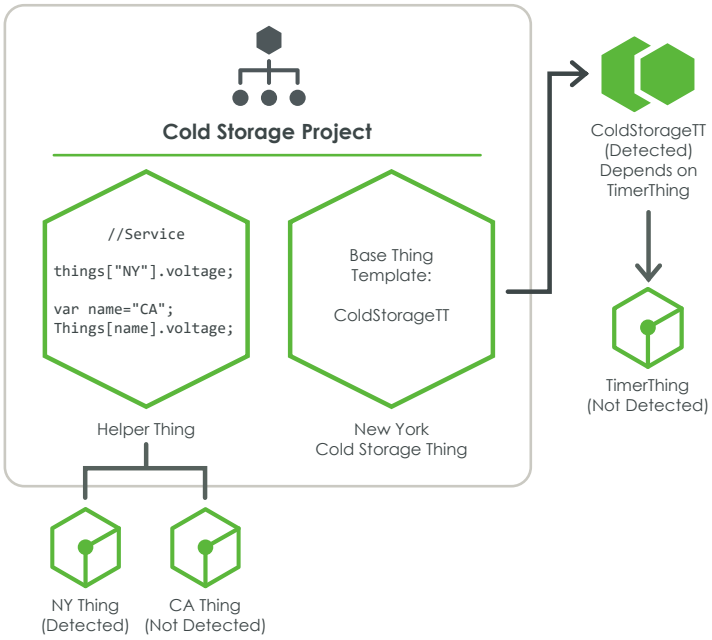
DATA MIGRATION

LISTEXTERNALDEPENDENCIES SERVICE

Every project has the ListExternalDependencies service. When run, it attempts to find missing dependent entities.

The scan is imperfect. It will not find:

- Entities indirectly referenced in the service/subscription code
- Dependencies more than one level deep



ptc university

Every project has the ListExternalDependencies service. When run, it attempts to find missing dependent entities.

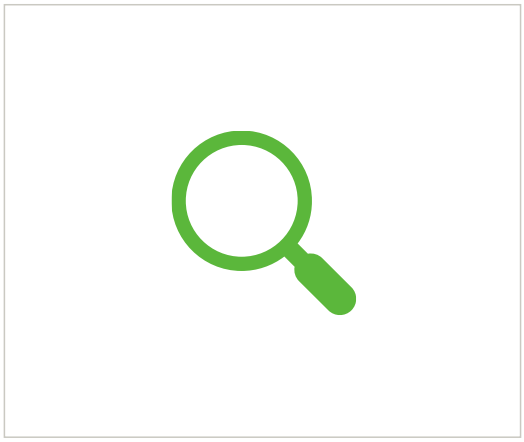
However, this service is not perfect, and it does not detect all dependencies. It does detect most of them, including permissions assigned to users and organizations that aren't included in the project.

The two most significant types of dependencies that it will not detect are:

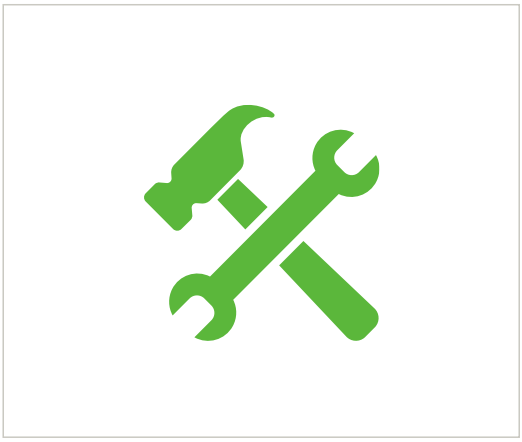
- Dependencies that are indirectly referenced in service or subscription code. In the Helper Thing service shown, the scanner service will detect the NY Thing as a dependency because it is referenced directly in the service code. However, it will not detect the CA Thing because it is indirectly referenced.
  Note: This project would import successfully but would cause a run time error when the service was executed, and the CA Thing was not found.
- Dependencies are only scanned one level deep. The New York Cold Storage Thing shown is dependent upon the ColdStorageTT Thing Template. The scanner would detect that dependency. However, let's suppose the ColdStorageTT has a subscription that listens for a timer event on TimerThing. The scanner would not detect TimerThing because the dependency is two levels deep.
  Note: This project would fail to import, as it couldn't import the ColdStorageTT Thing Template without its dependent TimerThing.

DATA MIGRATION

EXERCISE 3 FIND MISSING PROJECT DEPENDENCIES



1 Find Dependencies



2 Resolve Dependencies

ptc university

Note: Use the development database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Find Dependencies

1. Navigate to the **Services** page of the **PTC.CS.ColdStorage.PJ** project.
2. Execute the **ListExternalDependencies** service.

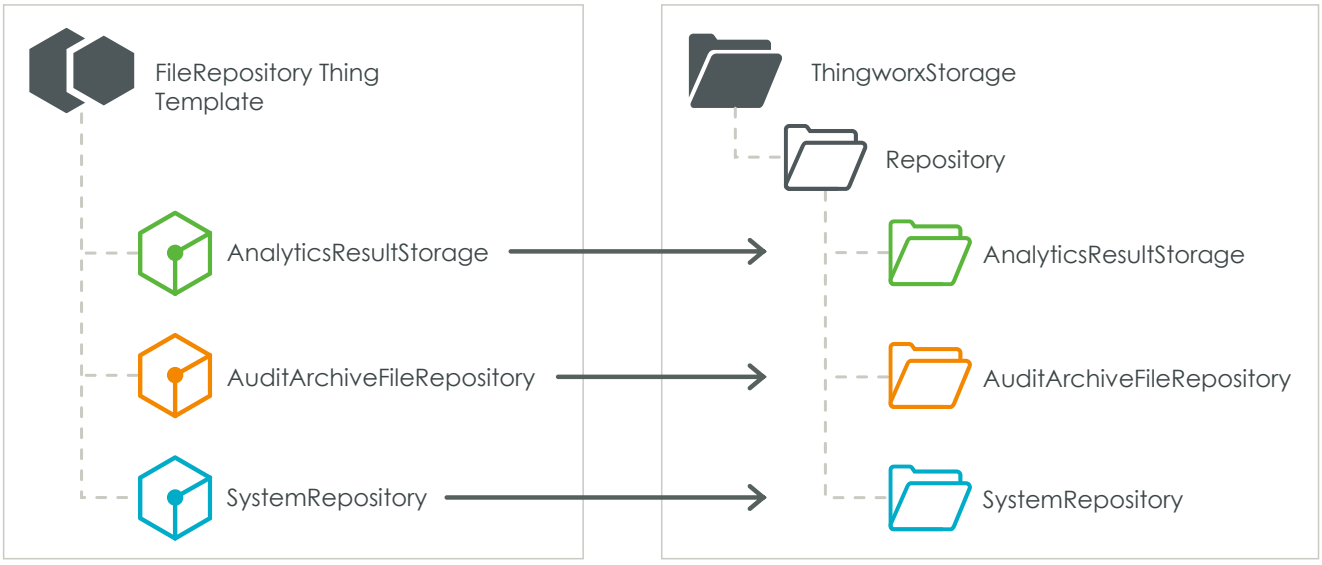
Task 2: Resolve Dependencies

Note: The PTC.CS.CreateColdStorageRoom.POP Mashup and the CSVParser extension are dependencies that could cause an import of the project to fail, so we will add them to the project.

1. On the Entities page, add the **PTC.CS.CreateColdStorageRoom.POP** Mashup to the PTC.CS.ColdStorage.PJ project.
2. On the project's General Information page, notice the **CSVParser_Extensions** extension as an extension dependency.
   Note: This only documents that the extension is needed for the project to work. It does not add the extension to the project.
3. Click **Save**.
4. Execute the **ListExternalDependencies** service again.
   Note: The New York user and group are not part of the application itself. They are test objects that you created when testing the application. To fix these, we remove the dependency instead of adding the project.
5. Navigate to the **Organization** page of the **PTC.CS.ColdStorage.Org** organization.
6. Remove the **NewYorkColdStorage** organizational unit.
   Note: Hover over the organizational unit, and then click the X.
7. Execute the **ListExternalDependencies** service again.
   Note: The only remaining dependency should be the CSV parser extension. Extensions cannot be added to the project, so this dependency cannot be removed.

DATA MIGRATION

FILE REPOSITORY



ptc university

In the previous exercise, data was not downloaded to your browser. Remember the timestamp from the ApplicationLog? Our exported data were placed in a file repository.

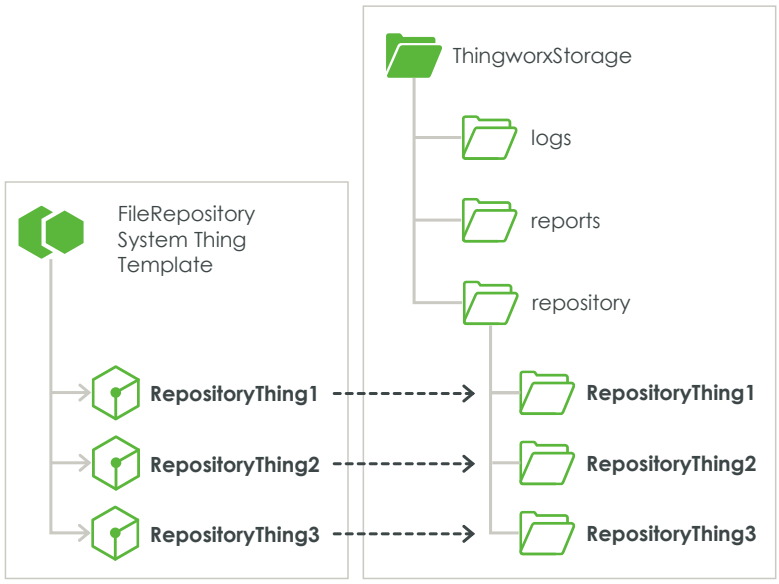
A file repository is a Thing derived from the FileRepository Thing Template. It maps directly to a folder on the ThingWorx server. The Thing name corresponds to the name of the folder underneath the ThingworxStorage/repository folder.

We send the data export to the SystemRepository folder in our exercise, which comes with ThingWorx by default. However, we could have created another file repository Thing, and sent the data export there as easily.

DATA MIGRATION

FILE REPOSITORY CONTENTS

A file repository entity is linked to a folder on the ThingWorx server's file system. When you export a file repository entity, it includes the entity and the link to the folder. However, it does not include the contents of that folder. If you need the files inside the repository on the target system, you need to copy them independently.

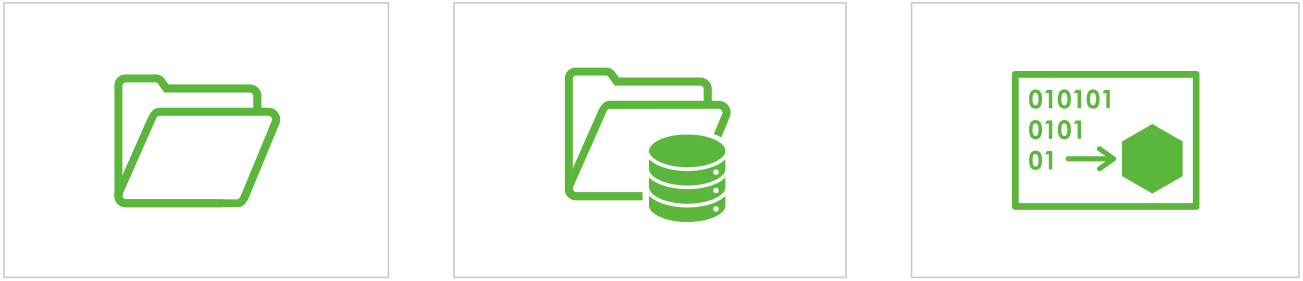


ptc university

A file repository entity is linked to a folder on the ThingWorx server's file system.

When you export a file repository entity, it includes the entity and the link to the folder. However, it does not include the contents of that folder. If you need the files inside the repository on the target system, you need to copy them independently.

EXERCISE 4 EXAMINE THE SYSTEM REPOSITORY



1 Examine the SystemRepository in Windows Explorer

2 Examine the SystemRepository in ThingWorx Composer

3 Download Value Stream Data

ptc university

Task 1: Examine the SystemRepository in Windows Explorer

1. Open Windows File Explorer.
2. Type **C:** in the navigation bar.
   Note: Your virtual software environment does not enable you to browse to the C:\ drive in File Explorer. However, you can access it by typing the path manually.
3. Navigate to C:\Program Files (x86)\ThingWorxFoundation\ThingworxStorage\repository\SystemRepository.
4. Open the folder that was created in the previous exercise when you exported ThingWorx data.
5. Open the **ValueStreams** folder.
6. Open the PTC.CS.ColdStorage.VS folder.
7. Note the **data-0.twx** file.

Task 2: Examine the SystemRepository in ThingWorx Composer

1. In the ThingWorx Composer, click **Manage**.
2. Select **SystemRepository.**
3. Navigate to the PTC.CS.ColdStorage.VS folder.

Task 3: Download Value Stream Data

1. Download **data-0.twx**.

DATA MIGRATION

COLLECTION PERMISSIONS

Collections are not entities, so they are not included in entity export. But they do have permissions that may be needed for your application to run correctly. There are a few ways to handle collection permissions:

- Use collection permissions sparingly
- Author setup service
- Export collection permissions independently

```
…

// Add Visibility to PTC.ColdStorage.Org on Resources collection
var params = {
    principal: "PTC.ColdStorage.Org" /* STRING */,
    principalType: "Organization" /* STRING */,
    collectionName: "Resources" /* STRING */
};
Resources["CollectionFunctions"].AddCollectionVisibilityPermission(
params);
logger.debug("Adding visibility permission for PTC.ColdStorage.Org
on Resources collection.");
…
```

ptc university

Collection permissions are not imported with entities because collections are not entities.

There are a few ways to handle collection permissions in your application.

- One is to avoid using collection permissions entirely. It is generally a good practice to use them sparingly.
- Another option is to author a setup script like the one shown. This setup script becomes part of the application migration process and sets up any configurations that are not handled by the import. It can set collection permissions, system user permissions, configure subsystems, or any other tasks that need to be done to finalize your application.
- The last option is to export collection permissions independently. The CollectionFunctions resource contains services to import and export collection permissions to a file. This file can only be exported and imported using the service – the main import/export menu does not create or import them.

Our ColdStorage application uses a setup service. It is the SetupCollectionAndSystemPermissions service on the PTC.CS.ColdStorage.Helper Thing.

DATA MIGRATION

BEST PRACTICE: USE COLLECTION PERMISSIONS SPARINGLY

Collection permissions do not just affect the project you are working on but all projects on the server.

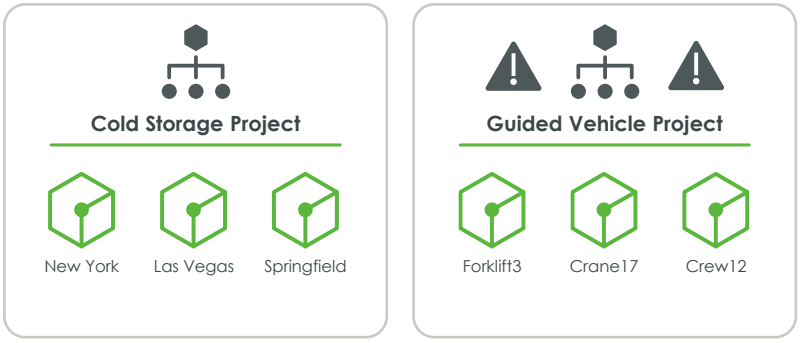
Good Uses:

- System user permissions
- Collections without sensitive information:
  - Media entities
  - Style or state definitions
  - Localization tables
  - Tag vocabularies

Insecure uses:

- Thing Templates
- Things
- Data tables/streams

```
// Visibility to PTC.ColdStorage.Org on Resources collection
var params = {
    principal: "ColdStorageAdministrator",
    principalType: "User",
    collectionName: "Things"
};
Resources["CollectionFunctions"].AddCollectionVisibilityPermission
(params);
```



ptc university

It is generally a best practice to use collection permissions sparingly and carefully.

Be cognizant that collection permissions do not just affect the project you are working on but all projects on the server. So, if you grant a collection permission to your users, those users will have permission to access that type of entity in other applications.

There are good uses for collection permissions that are generally safe:

- The system user generally has superuser-level access to all collections. This is acceptable from a security standpoint because the system user is only used when executing the service and subscription code you write and test. Nobody can log in as a system user remotely.
- There are collections that are visible to all users because they do not generally provide access to sensitive information. Examples are style and state definitions, media entities, and localization tables.

There are also inappropriate collection permissions. For example, if you grant permission to the data tables collection, that user can access all data tables across all applications on the server. This can easily create a security issue.

In the example, by granting the cold storage administrator visibility to the Things collection, they get visibility to everything in the guided vehicle project.

DATA MIGRATION

SYSTEM ENTITIES

Cannot be placed in project or tagged

May be exported independently of project/tag

Affect all applications on server

May include:

- Subsystem configurations
- SYSTEM user permissions
- System entity permissions

Ways to handle:

- Export needed system entities individually
- Author the Setup service

```
…
//Instance Visibility to GenericThing to
PTC.ColdStorage.Org.Administrators
ThingTemplates["GenericThing"].AddInstanceVisibilityPermission({
    principal: "PTC.ColdStorage.Org:Administrators" /* STRING */,
    principalType: "OrganizationalUnit" /* STRING */
});
logger.debug("Adding visibility instance permission for
PTC.ColdStorage.Org:Administrators on GenericThing Thing
Template.");
```

ptc university

System entities can't be exported as part of a project or tag because they can't be tagged or placed in a project. Examples include:

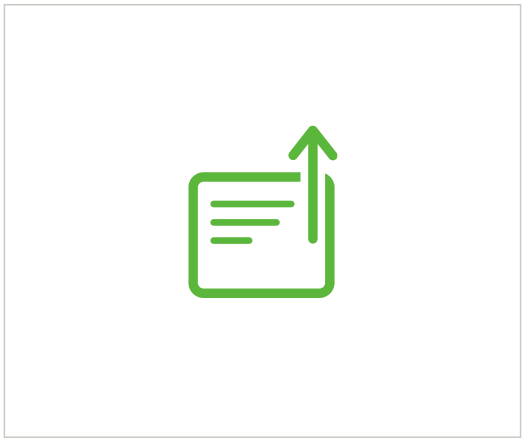
- Subsystems
- The SYSTEM user
- System-level entities, such as the GenericThing Thing Template

System entities can be exported independently of a tag or project – either individually, or as a collection, so there are two basic strategies to handling system entities:

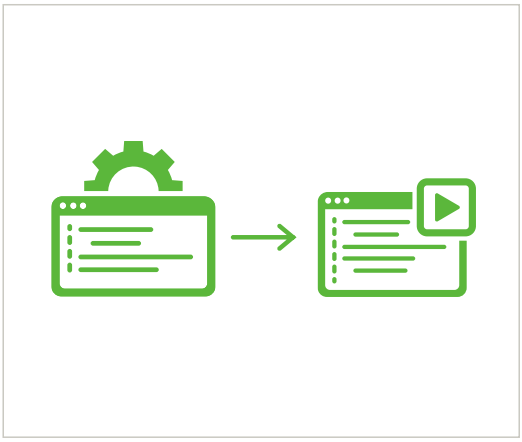
- You may export the entities you need individually. This may mean more than one export file and more than one file to import.
- You may configure system entities using a setup service like the one shown.

DATA MIGRATION

EXERCISE 5 EXPORT THE PTC.COLDSTORAGE.PJ PROJECT FROM DEVELOPMENT



1 Export the Project



2 Start the Test Database

ptc university

Note: Use the development database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Export the Project

1. Export with the following settings:
    - Export Option: **To File**
    - Export Type: **Collection of Entities**
    - Collection: **All**
    - Project: **PTC.CS.ColdStorage.PJ**
    - Export Format: **Binary**
    - Universal Export: **true/checked**
    - Include dependents of this project: **True/Checked**
2. Save the file as **W:\TWFD-DPLY-Lab-Files\DevExportEntity.twx**.

Task 2: Start the Test Database

1. Close the web browser.
2. Open Windows File Explorer. Then, double-click the **W:\TWFD-DPLY-Lab-Files\SwitchTWXInstance** shortcut.
3. Press **3** to use the test ThingWorx database.

DATA MIGRATION

SOLUTION PACKAGES

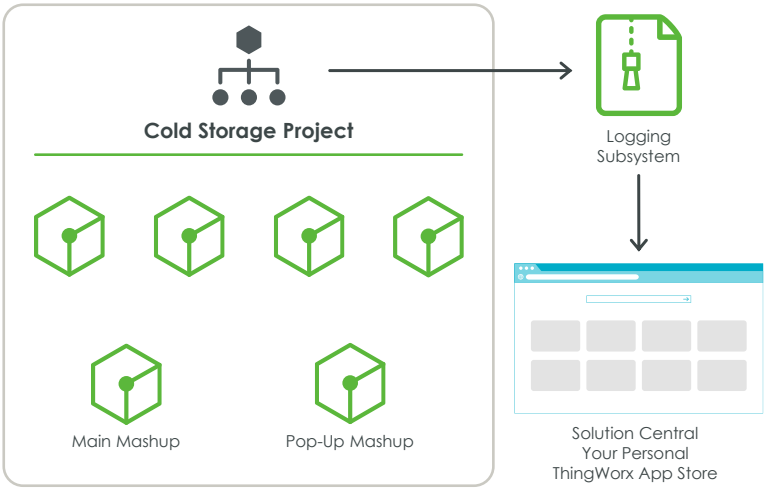
An alternative to exporting your project to a binary or XML file is to create a solution package.

If configured to connect to Solution Central, extensions:

- are uploaded to your Solution Central-based app store.
- are hosted by PTC.
- may be deployed over the Internet to other systems.

If not configured to connect to Solution Central:

- Zip file is placed in PackagedSolutions repository on the ThingWorx server.



ptc university

An alternative to exporting your project to a binary or XML file is to create a solution package available on the Package tab of the project in ThingWorx Composer.

This turns the entities of your project into an extension.

If ThingWorx is configured to connect to Solution Central, a PTC-hosted service, that package is uploaded to your Solution Central account. Then, it may be deployed to other ThingWorx servers.

If ThingWorx is not configured to connect to Solution Central, you can still create a solution package. The extension is placed in the PackagedSolutions repository on your ThingWorx server and may be imported like any other extension.

The extension contains the same entities as an export to project would and is subject to the same drawbacks, such as not having collection permissions or system entities.

DATA MIGRATION

MULTIPLE IMPORT FILES

An entity needs all dependent entities imported first, or an error occurs.

If you have multiple import files, you must use them to ensure dependencies are imported first.

Using the suggested order in the table will prevent dependency conflicts most of the time.

Example: Thing Template dependent on timer Thing. Timer Thing must be imported first.

Import Order

1. Extensions
2. Users, User Groups, and Organizations
3. Persistence Providers and Persistence Packages
4. Localization Tables and Model Tags
5. Media Entities, State Definitions, and Style Definitions
6. Application Keys and Authenticators
7. Data Shapes
8. Thing Shapes
9. Thing Templates
10. Subsystems
11. Things
12. Data

ptc university

When you use multiple import files, you are likely to run into dependency issues.

For an entity to be imported successfully, all entities it is dependent upon must already be in the ThingWorx system. This should not be an issue if you constructed your project correctly to include all required entities but is frequently an issue when dealing with smaller imports, such as tag-based exports, individual exports, or exports to source control.

This means you must import your files in the correct order. Using the order in the table will prevent dependency conflicts most of the time. If you look at the order, it makes sense. Every entity can have permission references to users, groups, and organizations, so those entities must be imported early. Most of the entities listed under Data Shapes can have infotables that use Data Shapes.

However, importing in this order does not guarantee there will be no dependency issues. For example, a Thing Template may have a subscription to a timer thing. If you try to import the Thing Template before importing the timer thing, it will fail.

DATA MIGRATION

DISCUSSION

What is the easiest way to avoid having conflicts between multiple import files?

ptc university

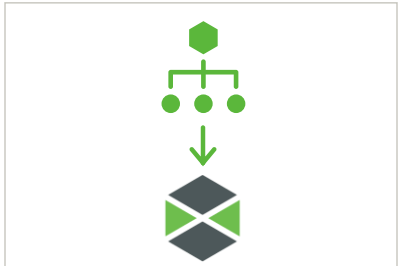
The easiest way is simple. Do not use multiple entity import files.

If you have a single, shared development environment, a single project, and thus only have one import file, conflicts will not happen.

If you use multiple development servers, these types of conflicts must be managed in your SCM system.

EXERCISE 6 IMPORT THE PROJECT TO TEST

1 Import Extensions



2 Import the Project Entities

3 Execute the Setup Service

ptc university

Note: Use the test database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Import Extensions

1. Launch and log on to ThingWorx with the username **Administrator** and the password **ptc-university-123**.
2. Import **W:\TWFD-DPLY-Lab-Files\1229-CSVParser_Extensions.zip**.
3. Verify the extension is imported under **Manage > Installed Extensions**.

Task 2: Import the Project Entities

1. Import with the following options:
   - Import Option: **From File**
   - Import Type: **Entity**
   - Import Source: **Single File**
   - File Name: **W:\TWFD-DPLY-Lab-Files\DevExportEntity.twx**

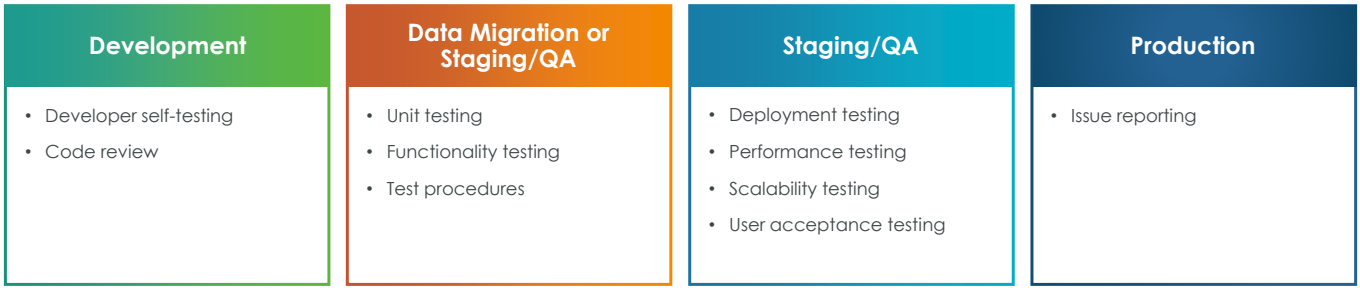
   Note: This is the file we exported in the last exercise
2. Wait for the Import successful message.

Task 3: Execute the Setup Service

1. Navigate to the **Monitoring > ScriptLog** page.
2. Click **Configure**.
3. Set the Log Level to **Debug**.
4. Navigate to the **Services** page of the **PTC.CS.ColdStorage.Helper** Thing.
5. Execute the **SetupCollectionAndSystemPermissions** service.
6. Navigate to the **Monitoring > Script Log** page.
7. Click **Refresh**.
8. Verify that debug messages exist, showing the permissions were added.

DATA MIGRATION

TEST TYPES



ptc university

There are many types of application testing done across the entire application development cycle. The earlier an issue is found, the more quickly and easily it is to fix.

During development, code reviews and developers testing their code as they build it can greatly reduce the number of issues that need to be managed later.

Functionality testing can begin once the application has made it to the Data Migration. This type of testing proves that the application works without regard to performance. Test procedures should be written and followed to verify that the application works as expected.

Since the QA server accomplishes two major goals:

- It tests the deployability of the application. If the application can be moved to QA successfully, it should migrate to production the same way.
- It tests several types of performance – responsiveness, scalability, and user acceptance.

By the time you reach the production server, the deployment instructions and testing should be complete.

DATA MIGRATION

TEST PROCEDURE DOCUMENTS

Services:

- Specify inputs and expected results
- Test expected and unexpected inputs

Test Procedure for CreateUserAndAssignUserGroup Service		
	Test Number 1	**Test Number 2**
Run Test As	ColdStorageAdministrator	ColdStorageAdministrator
Precondition	User Indianapolis does not exist	User Chicago does not exist
csName input	Indianapolis	Chicago
Password input	changeme123456	undefined
Expected result	Success (200 OK) • Indianapolis user created • Group PTC.CS.Indianapolis.Group created • User is member of group	Failure (500 Internal Server Error) • User not created • Group not created • Warning sent to script log

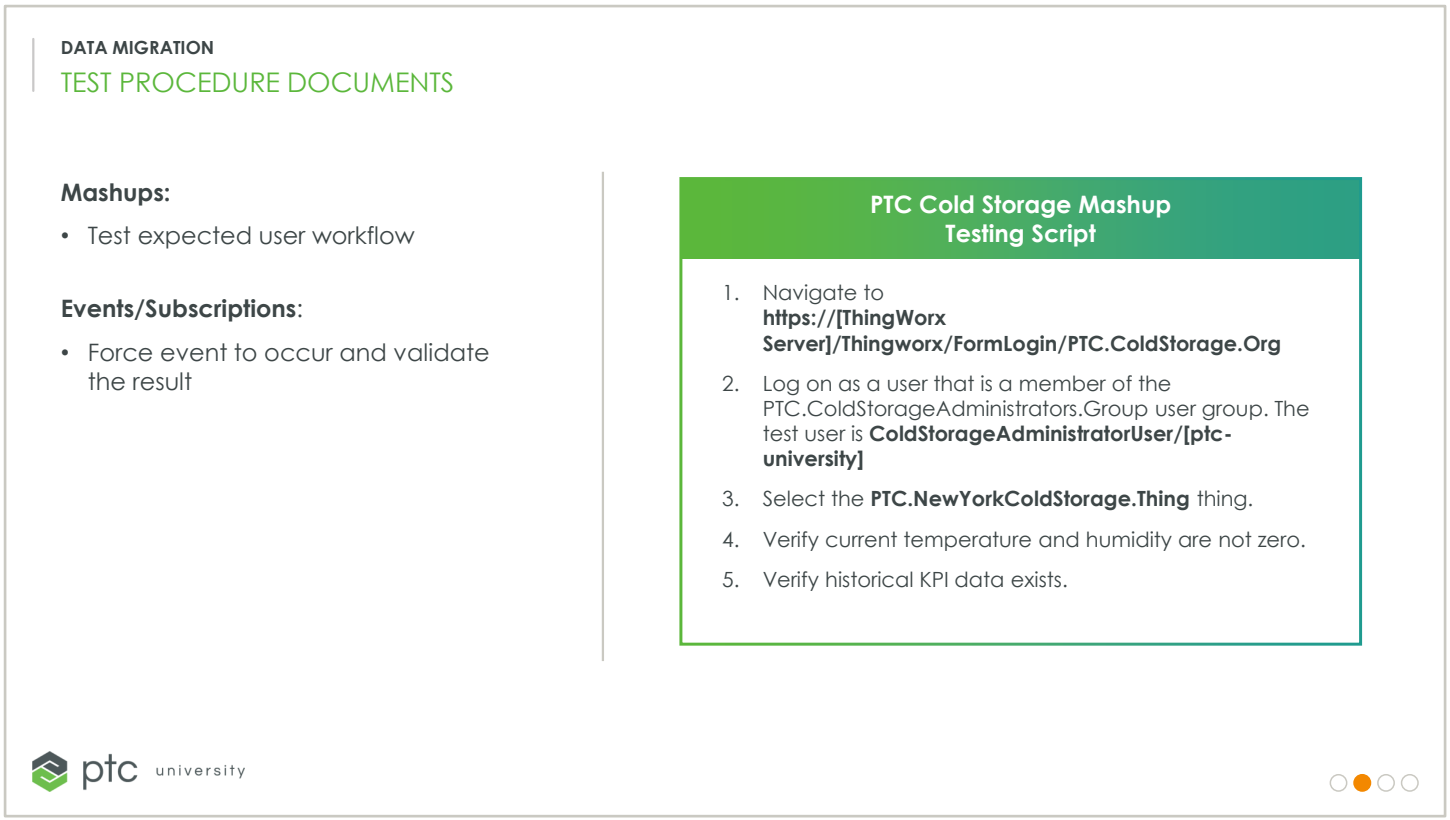
ptc university

Every service and Mashup should have a corresponding, documented test procedure. These can be written in any tool.

Service test documents should specify the inputs and expected outputs. They should not only test successful results but cases where there is unexpected input, such as test case 2, in the example, where the service is executed without a password.

Mashup test documents verify the user interface works. They read very much like the lab instructions in this course, telling the user where to click/type and what to look for.

Testing events and subscriptions are a bit more complex. You must have a way to force the event to occur. This may mean disconnecting the device and forcing the properties to change manually or connecting the server to a simulated data source that will force the event to occur.



Every service and Mashup should have a corresponding, documented test procedure. These can be written in any tool.

Service test documents should specify the inputs and expected outputs. They should not only test successful results but cases where there is unexpected input, such as test case 2, in the example, where the service is executed without a password.

Mashup test documents verify the user interface works. They read very much like the lab instructions in this course, telling the user where to click/type and what to look for.

Testing events and subscriptions are a bit more complex. You must have a way to force the event to occur. This may mean disconnecting the device and forcing the properties to change manually or connecting the server to a simulated data source that will force the event to occur.

DATA MIGRATION

GOOD PRACTICE: TEST AUTOMATION TOOLS

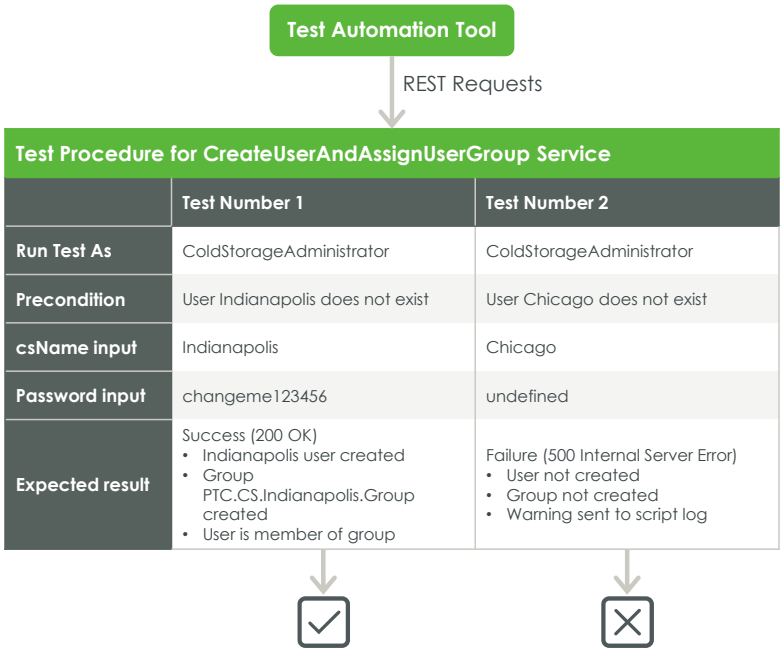
It is an excellent practice to test services with test automation tools.

These tools:

- Send REST requests in a batch
- Validate results of REST requests
- Measure performance

Some test automation tools are:

- Apache Jmeter
- SoapUI
- Rest-Assured



ptc university

In the testing exercise for this course, we are going to use Postman for two reasons:
- It was presented in prerequisite courses, so everyone taking this class should know how to use it.
- It presents how to set up REST requests to test services.

However, Postman is not the best tool for testing services. A test automation tool, such as Jmeter or SoapUI, is a much better choice. These tools enable you to script a large number of REST requests following your test procedure document.

Any test automation tool that can send scripted REST requests will work with ThingWorx.

DATA MIGRATION

ALTERNATIVE PRACTICE: CREATE TEST MASHUP

Simple Mashups that execute service and get the result.

Does not display 200 OK status code.

Do not package test Mashups with project.

Test Procedure for CreateUserAndAssignUserGroup Service

	Test Number 1	Test Number 2
Run Test As	ColdStorageAdministrator	ColdStorageAdministrator
Precondition	User Indianapolis does not exist	User Chicago does not exist
csName input	Indianapolis	Chicago
Password input	changeme123456	undefined
Expected result	Success (200 OK) • Indianapolis user created • Group PTC.CS.Indianapolis.Group created • User is member of group	Failure (500 Internal Server Error) • User not created • Group not created • Warning sent to script log

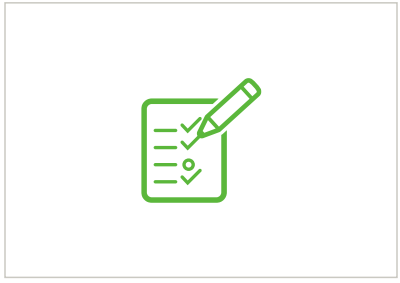
ptc university

Another way to test services is to create testing Mashups. These Mashups execute the service with user-provided inputs. Then, follow the test procedure and verify the results.

These Mashups should not be packaged with the project to be sent to the production server. To work, you must give ordinary users permission to run the Mashup, but you do not want to provide this type of access to individual services on the production server.

EXERCISE 7 TEST THE CREATEUSERANDASSIGNUSERGROUP SERVICE

1 Fix Standards Issues with the Service



2 Test the CreateUserandAssignUserGroup Service

ptc university

Note: Use the test database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Fix Standards Issues with the Service

1. Navigate to the **Services** page of the **PTC.CS.ColdStorage.Helper** Thing.
2. Examine the code for the **CreateUserAndAssignUserGroup** service. Successful entity creation should be logged at the DEBUG level, not the WARN level, and should include the username.
   Change:
   ```
   logger.warn(userName + " user created.");
   to
   logger.debug(userName + " user created.");
   ```
   Note: It is easier to read the code if you maximize it and collapse the left explorer bar.
3. Click **Done** and **Save**.

Task 2: Test the CreateUserandAssignUserGroup Service

1. Open the **PTC.CS.ColdStorage.Helper** Thing and navigate to the Service menu.
2. Click the **CreateUserandAssignUserGroup** Service.
3. Enter the following inputs:
   userName: **Indianapolis**
   password: **changeme123456**
4. Click **Save** and **Execute.**
5. In ThingWorx, navigate to the **Members** page of the **PTC.CS.Indianapolis.Group** entity.
6. Verify the Indianapolis user is a member.

SECTION OVERVIEW

1 DEVELOPMENT

2 DATA MIGRATION

3 QUALITY ASSURANCE/STAGING

4 LOGS AND REPORTS

5 PRODUCTION

ptc university

QUALITY ASSURANCE/STAGING

HIGHLIGHTS

This section focuses on the Quality Assurance staging server. We will look at the function of this server and which servers the data and application are imported to the QA server from.

ptc university

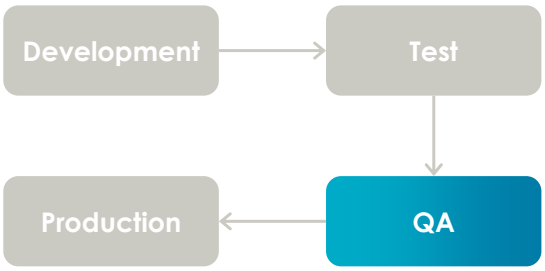
QUALITY ASSURANCE/STAGING

QUALITY ASSURANCE STAGING SERVER

The quality assurance server is used to validate deployment procedures. This is where the application is formally tested, including scalability, user acceptance, and performance testing.

The quality assurance server should mimic a production server as closely as possible:

- No application changes in QA
- Similar hardware, clustering, and database
- Realistic data
- Realistic connections



ptc university

The quality assurance server is a critical stage in the migration from development to production.

This is where the application is formally tested, including scalability, user acceptance, and performance testing. The QA server also helps formalize the strategy for migrating to the production server.

The QA server should mimic the production server as closely as possible, so the performance and user experience are similar to that of the production server.

QUALITY ASSURANCE/STAGING

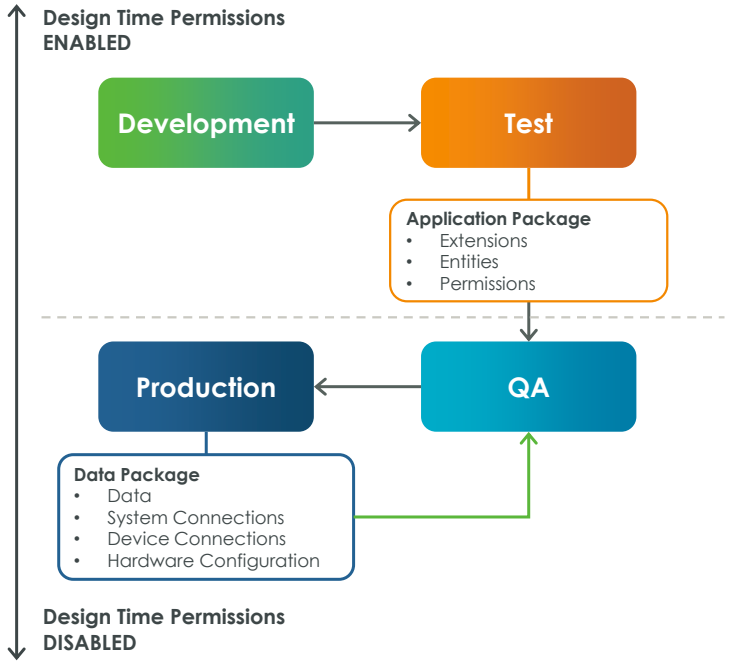
QUALITY ASSURANCE STAGING SERVER

When building the QA server, the primary gold is to mimic the production server as closely as possible:

- Data
- System connections/integrations
- Device connections
- Hardware and database

The application is imported from the test server. An application package should be created as a base point of:

- Extensions
- Entities
- Permissions



ptc university

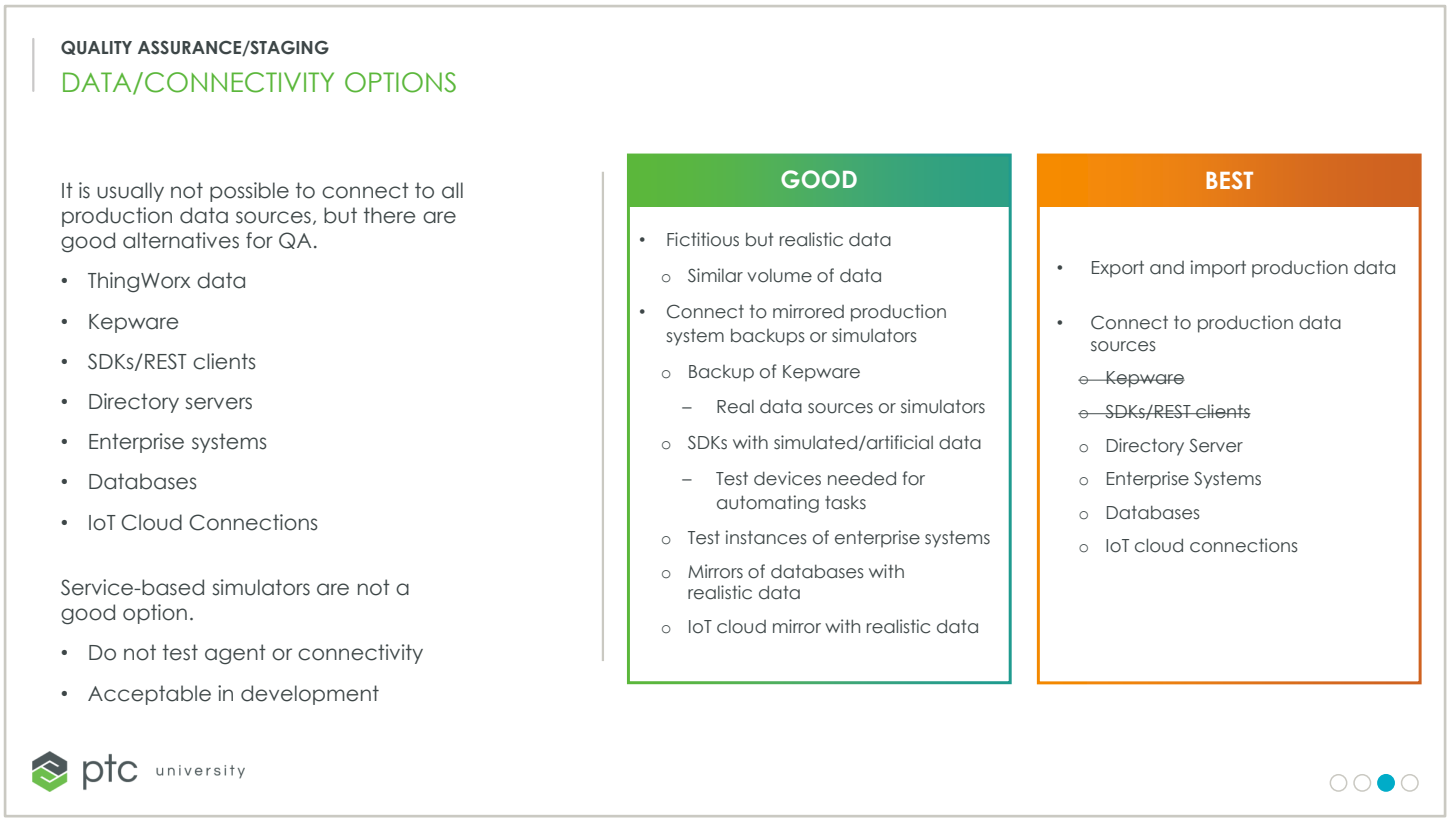
When building the QA server, the primary goal is to make it as realistic as possible to duplicate the production server.

- Data in streams and data tables should be exported from production.
- System connections should be to production servers:
  - Production directory server handles user logins.
  - Production ERP, databases, or any other systems the application requires.
- Device connections should be to real production devices
- QA hardware and platform should be the same as the production server

An exact duplicate of the production server may not be possible due to financial constraints or security concerns, but the closer you are to a real production system, the more realistic and valid your testing will be.

While data and connections come from the production server, the application itself comes from the test server.

An application package should be created, which is a base point containing the application extensions, entities, and permissions. This is everything the application needs to run.



In most cases, you can't create an exact mirror of the production server for QA. The problems may be technological, financial, or security reasons. A common reason is that the data on production systems is sensitive, and you do not want to make it available to your developers and testers.

There are alternatives to using production data that will enable you to engage in an adequate QA process. These options depend upon the type of data or connection we are dealing with.

- ThingWorx data: If you can't use production data for your streams and data tables, create a realistic data set. This data set should have a similar amount of data and a similar data profile. Be aware that artificial data is rarely adequate for use cases involving ThingWorx Analytics, as analytics requires a real data set to get real results.
- Kepware Server: You almost always have to mirror your Kepware Server. Kepware can only be configured to directly send AlwaysOn data to one ThingWorx server at a time, so if you want to connect it to QA, you have to disconnect it from production, which is rarely an option. However, Kepware can record real production device data to a simulator file and then play it back later. This is an excellent option for QA. Instead of connecting your QA instance of Kepware to production data sources, have it playback recorded data from production.
- SDK-based and REST clients: Like Kepware, these are usually configured to connect to one and only one ThingWorx server. You will probably need to create simulator clients, which connect to your QA instance of ThingWorx and send artificial data without being connected to a device. The artificial data should be as realistic as possible. It should contain all properties, services, and events of the production agents. Optimally, it should send data that was recorded from a production device. An extra problem arises with automation, where ThingWorx executes remote services. To verify those services work on a real device, you will need at least one real device with the production agent, not a simulated one.
- Directory servers: It is highly likely that you will be able to use a production directory server for QA purposes. Directory servers are designed to support multiple applications so that they can support production and QA simultaneously. If you can't use a production directory server, use a similarly configured test instance with similar data.
- Enterprise systems: There isn't usually a technological restriction to connecting to ERP, CRM, and other enterprise systems. However, there is frequently a security concern. You can't allow your QA system to make

changes in your production ERP system. The good news is that most organizations that have these systems also maintain a test instance of them so that you can connect to the test instance. Optimally, the instance you connect to should have realistic data.

- Databases: Databases are easy to mirror, or if there is production data, to create a simulated data set.
- IoT Clouds: Most IoT clouds run on protocols like OPC and MQTT that have no issue with multiple servers connecting to them, so it is fairly likely that you will be able to connect to production data. If you can't, you will need to mirror it. Fortunately, most IoT clouds are cloud-hosted, so replication is easy.

QUALITY ASSURANCE/STAGING

PROVISIONED ENTITIES

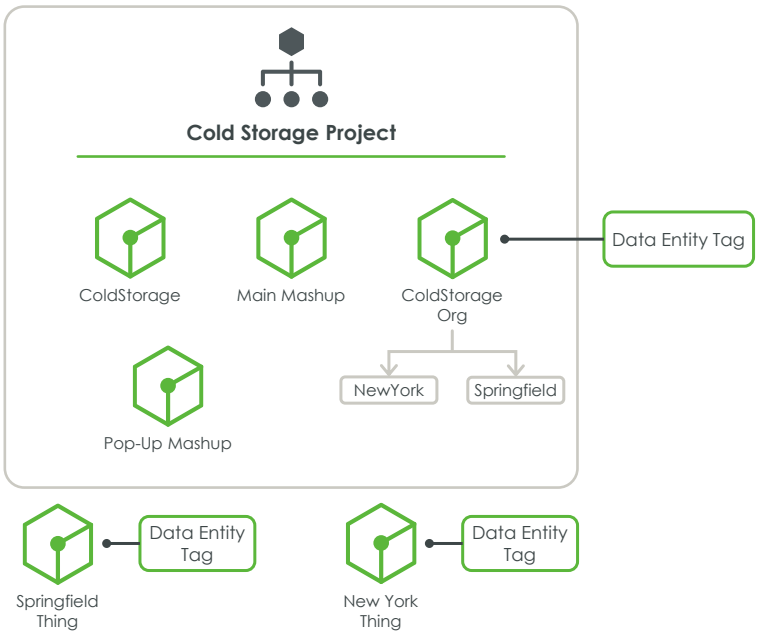
Entities are provisioned and altered as applications are used in production.

Cold Storage Things:

- Should not be part of project.
- Export from production and import separately.

Cold Storage Organizational Units:

- Organizational units are not entities.
- Export from development/test as part of project.
- Export from production separately. Production import overwrites organization.



ptc university

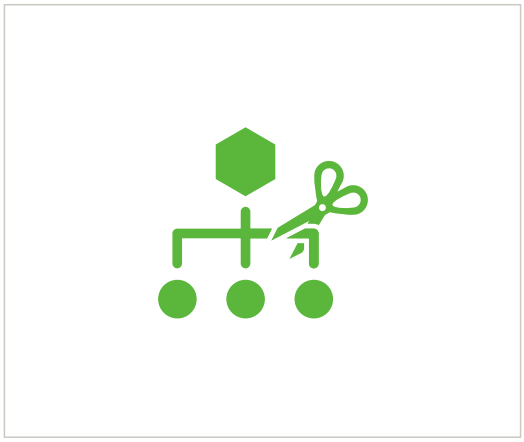
Some entities are production data. These are generally entities that are created or modified as the ThingWorx application is used. Let's look at our cold storage application and determine what entities should be treated as data.

Cold storage Things are created while running the Mashups. These entities are production data and should not be included in the project and should be exported from the production system. The easiest way to collect these types of entities is by using a model tag. Our application uses the Applications:DataEntities tag.

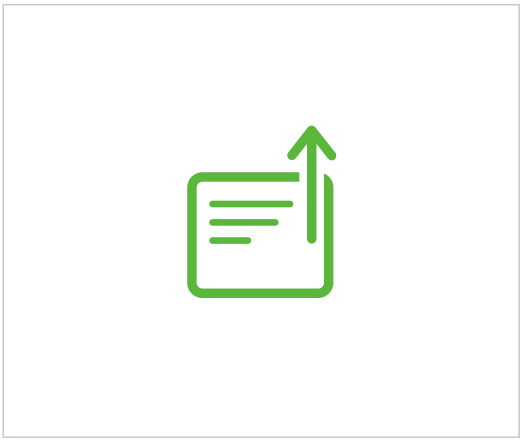
Our application also modifies the organization by adding organizational units. Organizational units are not entities, so they cannot be exported separately. The way to handle this is to export and import the organization twice:

- The first time, from development, as part of the project. It is imported with the product, so the rest of the entities in the application can refer to the organization for access control purposes.
- The second time, from production, as part of the data. This organization will contain additional organizational units. When imported, this will overwrite the development organization, creating the new organization units.

EXERCISE 8 EXPORT THE PTC.COLDSTORAGE.PJ PROJECT FROM TEST



1 Remove the Simulator from the Project



2 Export the Project

ptc university

Note: Use the test database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Remove the Simulator from the Project

Note: In development and test, our entities got property values from a simulator timer Thing. In QA, the simulator should be removed, and more realistic data sources should be used.

1. Navigate to the **General Information** page of the **PTC.CS.Simulator.Timer** entity.
2. Change the project to **PTCDefaultProject**.
3. Click **Save.**

Task 2: Export the Project

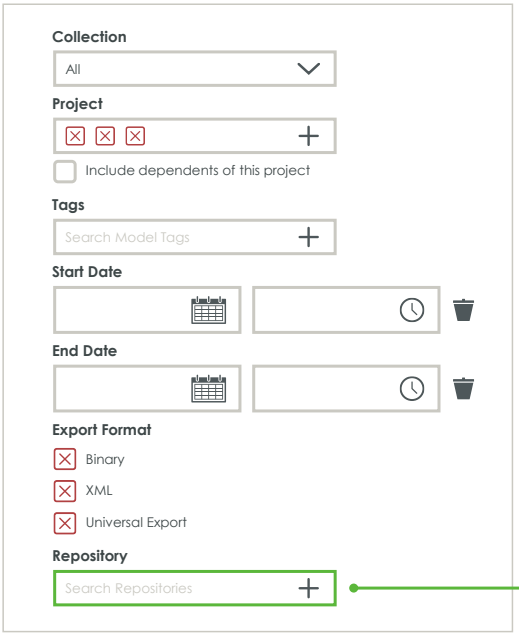
1. Navigate to the **Services** page of the **PTC.CS.ColdStorage.PJ** project.
2. Execute the **ListExternalDependencies** service.
3. Validate that the CSVParser extension is the only dependency.
4. Export with the following settings:
   - Export Option: **To File**
   - Export Type: **Collection of Entities**
   - Collection: **All**
   - Project: **PTC.CS.ColdStorage.PJ**
   - Include dependents of this project: **true/checked**
   - Export Format: **XML**
   - Universal Export: **true/checked**
5. Save the file as **W:\TWFD-DPLY-Lab-Files\TestEntityExport.xml**.
6. Close the web browser.
7. Open Windows File Explorer. Then, double-click the **W:\TWFD-DPLY-Lab-Files\SwitchTWXInstance** shortcut.
8. Press **4** to use the ThingWorx QA database.

QUALITY ASSURANCE/STAGING

EXPORT DATA FROM THINGWORX

Exporting data from ThingWorx is like exporting entities except:

- Cannot filter export on project
- Export format is always binary
- Must export to repository or ThingWorx storage
- Cannot export to client file



ptc university

Exporting Data from ThingWorx is very similar to exporting entities. They both use the same user interface, just with different options.

However, there are a few differences:

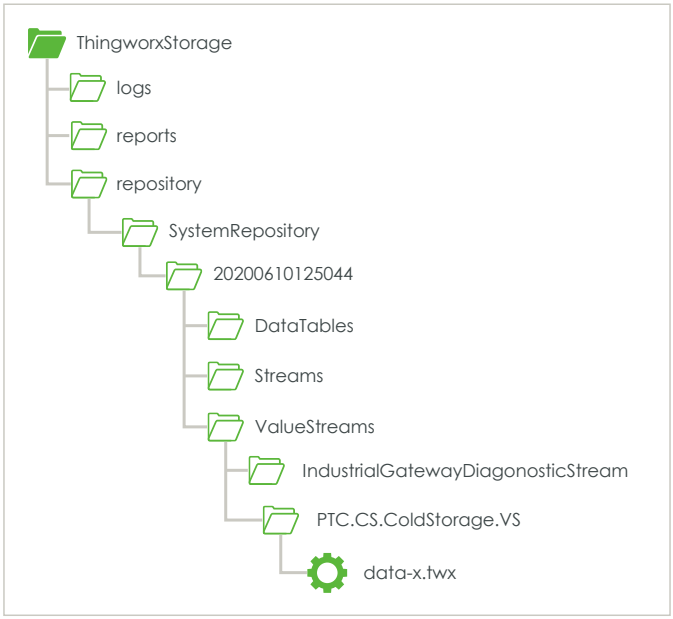
- You cannot filter a data export based on the project, but you can filter it based on tags.
- The export format is always binary. There are no XML data exports.
- You must export to a repository or ThingWorx storage. The data export will not be sent to the client. This is because data exports tend to be very large, and browser transfers are not reliable or efficient for large amounts of data.

QUALITY ASSURANCE/STAGING

DATA EXPORT LOCATION

The entire application's data files cannot be combined into a single file. Each entity has its own data file.

- Repository
- Timestamp
- Entity type
- Individual entity
- Data file



ptc university

Also, unlike entities, the entire application's data files cannot be combined into a single file.

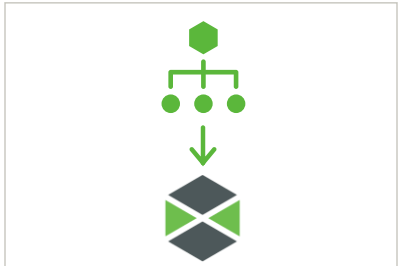
Multiple data export creates a directory structure with a distinct directory for each entity.

The top-level folder for the structure is the timestamp. Under that are folders for each entity type, folders for each entity, and lastly, the data export file for that individual entity.

QUALITY ASSURANCE/STAGING

EXERCISE 9 IMPORT TO QA

1 Import Extensions



2 Import the Project Entities

3 Execute the Setup Service

ptc university

Note: Use the QA database of ThingWorx for this exercise and log on as Administrator/ptc-university-123.

Task 1: Import Extensions

1. Launch and log on to ThingWorx with the username **Administrator** and the password **ptc-university-123**.
2. Import **W:\TWFD-DPLY-Lab-Files\1229-CSVParser_Extensions.zip**.
3. Verify the extension is imported under **Manage > Installed Extensions**.

Task 2: Import the Project Entities

1. Import with the following options:
    - Import Option: **From File**
    - Import Type: **Entity**
    - Import Source: **Single File**
    - File Name: **TestEntityExport.xml**
2. Wait for the Import successful message.

Task 3: Execute the Setup Service

1. Navigate to the **Services** page of the **PTC.CS.ColdStorage.Helper** Thing.
2. Execute the **SetupCollectionAndSystemPermissions** service.

EXERCISE 9 IMPORT TO QA

4 View the Mashup

5 Import the Data

6 Examine the Result

Task 4: View the Mashup

1. Open an incognito Chrome window.
2. Browse to **http://localhost:8181/Thingworx/FormLogin/PTC.CS.ColdStorage.Org**
3. Log on with the username **ColdStorageAdministrator** and the password **ptc-university-123**.
   Note: No data will be shown because no cold storage sites have been loaded.

Task 5: Import the Data

1. Return to Composer where you're logged in as Administrator
2. Import with the following options:
   - Import Option: **From File**
   - Import Type: **Entity**
   - Import Source: **Single File**
   - File Name: **W:\TWFD-DPLY-Lab-Files\TWDP-DPLY-Data.xml**
3. Refresh the incognito window.
   Note: Several cold storage sites have been loaded from a production export, but they have no data. Moreover, the data is not changing because the simulator was removed, and device agents have not been connected.
4. Import with the following options:
   - Import Option: **From File**
   - Import Type: **Data**
   - Import Source: **File Repository**
   - File Repository: **SystemRepository**
   - Path: **/20200616152804**
     Note: You may use the browse button to get the folder.

Task 6: Examine the Result

1. Navigate to the **Monitoring > ApplicationLog** page.
2. If you do not see the message indicating the data log import for PTC.CS.ColdStorage.VS has completed, wait a minute or two, and refresh the log.
   *Note: The log entry should be similar to **Task-1592331599076 Import SUCCESSFUL for entity [PTC.CS.ColdStorage.VS] file:[data-0.twx]. Total [3515] rows imported**.*
3. Refresh the incognito window. Notice that historical KPI data for temperature and humidity exists.

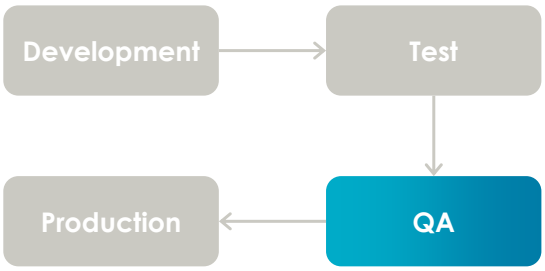
QUALITY ASSURANCE/STAGING

QA SERVER ACTIVITIES

The QA server is built for a wide variety of formal testing activities.

- Deployment testing complete!
- Customer acceptance testing
- Scalability/Performance testing:
  - Automated tools
  - Metrics
- Device compatibility testing

It is best practice to use a formal change control system to track and deal with application changes.



ptc university

The QA server is built for a wide variety of formal testing activities.

First, by creating the QA server, you have shown that the application is deployable, completing deployment testing.

However, there are many other types of testing to complete, a few of which include:

- Customer Acceptance Testing typically involves showing users the application and verifying they can use it with the help and documentation provided.
- Scalability and Performance Testing are critical before moving to production and are usually completed using automated testing tools such as Apache JMeter and SoapUI by Smartbear.
- Device Compatibility Testing is critical, as developers do not typically test every supported browser and device combination. Although using supported ThingWorx Mashup and widgets guarantee the Mashup will work on all ThingWorx supported browsers, you want to verify that the Mashup is functional at all supported screen resolutions without buttons that are too small to click or errant scrollbars.

Also, remember, it is a best practice not to introduce application changes on the QA server. Issues should be tracked and dealt with within a formal change control system, such as JIRA, Bugzilla, or Windchill PDMLink.

SECTION OVERVIEW

1 DEVELOPMENT

2 DATA MIGRATION

3 QUALITY ASSURANCE/STAGING

4 LOGS AND REPORTS

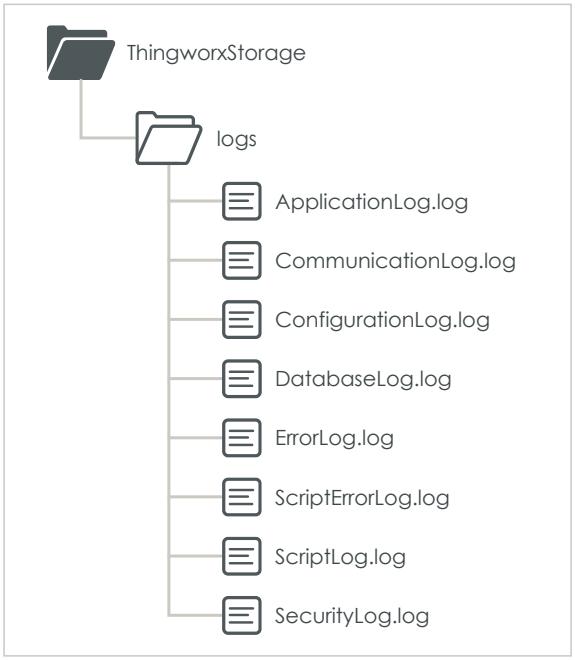
5 PRODUCTION

ptc university

LOGS AND REPORTS

SYSTEM LOGS

System Logs are stored in /ThingworxStorage/Logs and ThingWorx Composer.



ptc university

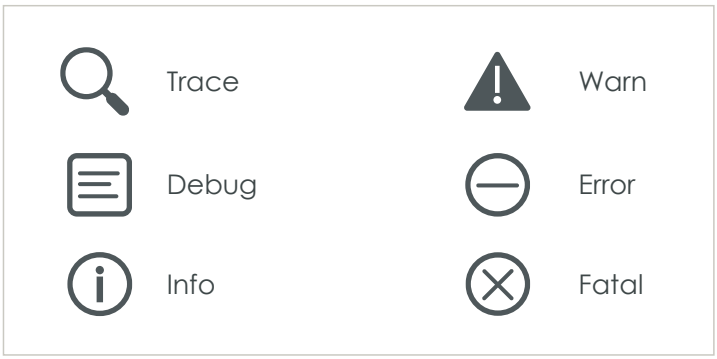
We have been looking at logs throughout this course. In this section, we'll go into more detail about the logs and reports that are available.

ThingWorx logs are stored in the /ThingworxStorage/Logs directory and accessed directly from the server file system or ThingWorx Composer.

LOGS AND REPORTS

LOG4J STANDARD

ThingWorx logs are recorded using the Log4j standard. Log4j standard is readable and filterable with many third-party log viewers.




```
2018-09-27 00:00:00.392+0000 [L: WARN] [O: S.c.t.d.e.DSLScript] [I: ] [U: testUser]
2018-09-27 00:00:00.392+0000 [L: WARN] [O: S.c.t.d.e.DSLScript] [I: ] [U: testUser]
2018-09-27 00:00:00.392+0000 [L: ERROR] [O: S.c.t.d.e.DSLScript] [I: ] [U: Shipyard_testUser]
2018-09-27 00:00:00.392+0000 [L: WARN] [O: S.c.t.d.e.DSLScript] [I: ] [U: testUser]
```

ptc university

ThingWorx logs are recorded using the Log4j standard. There are numerous log viewer applications for Log4j-based logs, including one that comes as part of ThingWorx Composer.

Log4j log entries have log levels, which describe how important the entry is. The log levels are FATAL, ERROR, WARN, INFO, DEBUG, and TRACE.

Generally, ERROR and WARN log entries contain valuable information to an administrator in a smoothly-running production server. Anything below that is for troubleshooting or development.

CONFIGURE LOGS

Using the ThingWorx log viewer, you can configure logs or filter them. When you configure a log, you prevent entries below a certain level from being entered into the log file.

ptc university

Using the ThingWorx log viewer, we can configure logs or filter them.

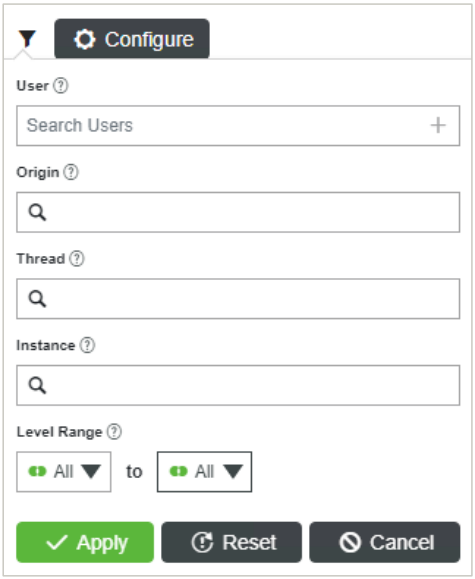
When you configure a log, you prevent entries below a certain level from being entered into the log file. So, if you configure your logs to only show entries at level WARN or higher, new entries at the DEBUG level won't be logged. Any information those entries would have contained is lost forever.

This isn't a bad thing – in a production system, low-level log entries are rarely accessed, and the processing and storage resources needed to create those log entries are significant. Generally, a good practice is to log at the WARN level in production, temporarily lowering the level for troubleshooting purposes.

LOGS AND REPORTS

FILTER LOGS

Filtering a log leaves entries in the log file but only displays a subset of entries in the view. This will help limit the number of log entries you see.



ptc university

The other way to reduce the number of log entries you see is to filter logs.

Filtering a log is done on the client, not the server, and doesn't affect the log file itself.

Instead, it just restricts the entries of the log file that you see in the client.

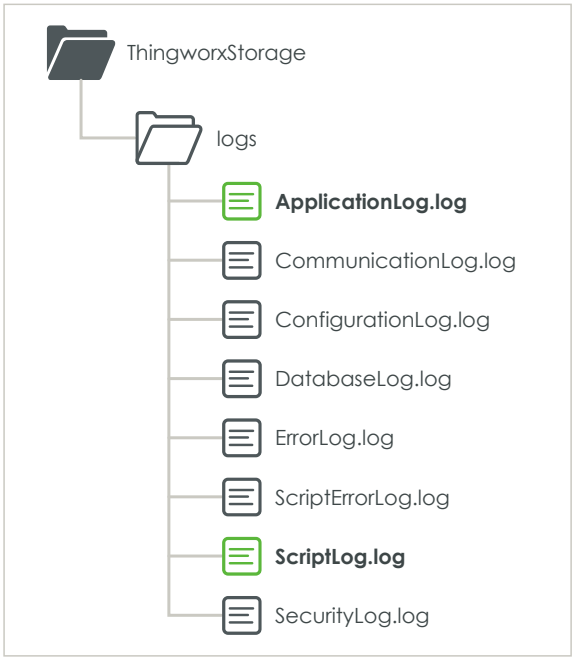
This can be more flexible – instead of just filtering on log level, you could ask for entries from a specific user or entity. This is very helpful when troubleshooting.

However, filtering a log has none of the performance advantages that configuring a log has. The server is still spending processing and storage resources for log entries, even if you do not view them in your client.

LOGS AND REPORTS

SYSTEM LOGS

The two logs you will look at the most are the application and script logs. The application log contains messages and issues from the ThingWorx server itself. The script log contains messages coming from services and subscriptions.



ptc university

The two logs you will look at the most are the application and script logs.

The application log contains messages and issues from the ThingWorx server itself. You have been examining this log throughout the course to uncover permission issues.

The script log contains messages coming from services and subscriptions – the code that you write in your application, while the script error log contains stack traces for errors in your code.

The other log files are for specific purposes, such as database issues or entity alterations.

SECTION OVERVIEW

1 DEVELOPMENT

2 DATA MIGRATION

3 QUALITY ASSURANCE/STAGING

4 LOGS AND REPORTS

5 PRODUCTION

ptc university

PRODUCTION

DEPLOYMENT TO PRODUCTION

If you deploy to QA and complete all of your QA tests properly, deploying it to production should go smoothly. Once in production, it is best to have an issue-tracking system for reporting issues and enhancement requests.

ptc university

If you deploy to QA and complete all of your QA tests properly, deploying it to production should be simple. Just repeat the procedure you established for deploying to QA, only connecting the real data instead of simulated data.

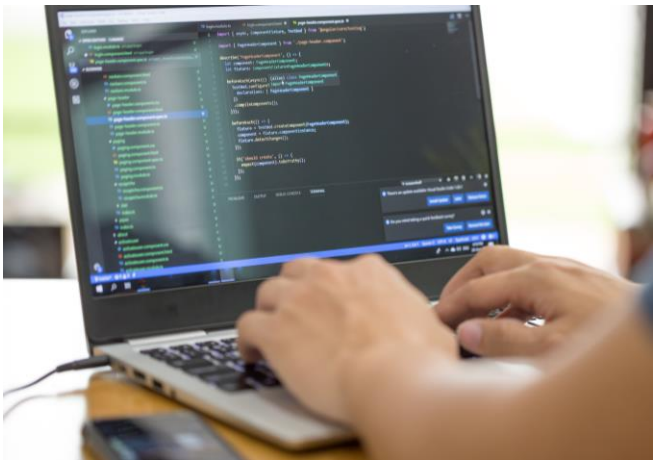
Once in production, it is best to have an issue tracking system for reporting issues and enhancement requests, such as JIRA or Bugzilla.

CONCLUSION

CONCLUSION

RECAP: BEST PRACTICE ROUNDUP

- Establish four servers
- Mashup-centric development
- Project contains application entities
- Use change control system in QA and production
- Copy entities before editing
- Establish coding conventions and standards
- Catch errors and handle unexpected output consistently
- Test as a non-administrative user
- Do code reviews
- Use test automation tools or test Mashups



ptc university

That is it for the course. You should now have a solid understanding of how to take a ThingWorx application from the development environment into production and follow important best practices along the way.

Questions and Answers

ptc university

ADDITIONAL RESOURCES

FOLLOW-UP RESOURCES

- Browse the PTC University Website
- Discover our courseware Curriculum
- Earn a Certification:
  - Fundamentals
  - Professional
- Check out the PTC Community
- Follow us on our social channels @PTC_University

ptc university

KNOWLEDGE CHECK ANSWERS

ptc university

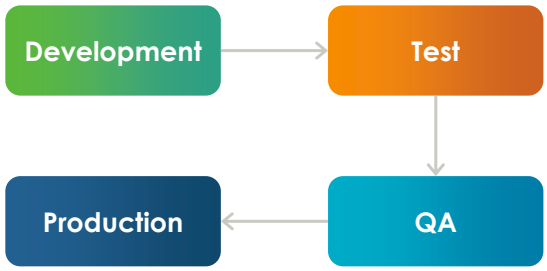
COURSE INTRODUCTION

KNOWLEDGE CHECK

1

On which server(s) should application issues not be corrected but reported to a change control system instead? *Select all that apply.*

A. Development
B. Test
C. QA
D. Production



ptc university

Knowledge Check 1 Answer

C and D. Changes in development and test may be made informally, but the QA and Production servers should only have a carefully controlled release.


ptc university