Lab: Organize builds with folders

Folders can be created to provide separate namespaces for different projects or teams on the same controller. Each folder is separate from other folders, so the "*build*" job in Folder A is distinct from "*build*" job in Folder B. Using folders simplifies branch and Pipeline management.

Properties and resources can be defined that are only visible to builds in that folder. For example, if you assign credentials to a folder, only Pipelines in that folder can use those credentials.

Folders are useful when seeding new projects. You can create a new folder and move existing Pipelines into that folder or you can clone an existing folder, leaving the children intact.

Folders can be nested like file system folders:

- Folder 1
  - Folder 2
    - Folder 3 … Folder *n*

The goal of this exercise is to use folders on the Jenkins instance by doing the following:

- Create a new folder and move an existing Pipeline into it.
- Clone that folder to create a new folder that includes the same Pipeline.
- Run the Pipelines in each folder and see that they are totally independent of each other.

Your group of developers will be split in two teams: **TeamA** and **TeamB**. They will work on the same application, but adapt it for different customers.

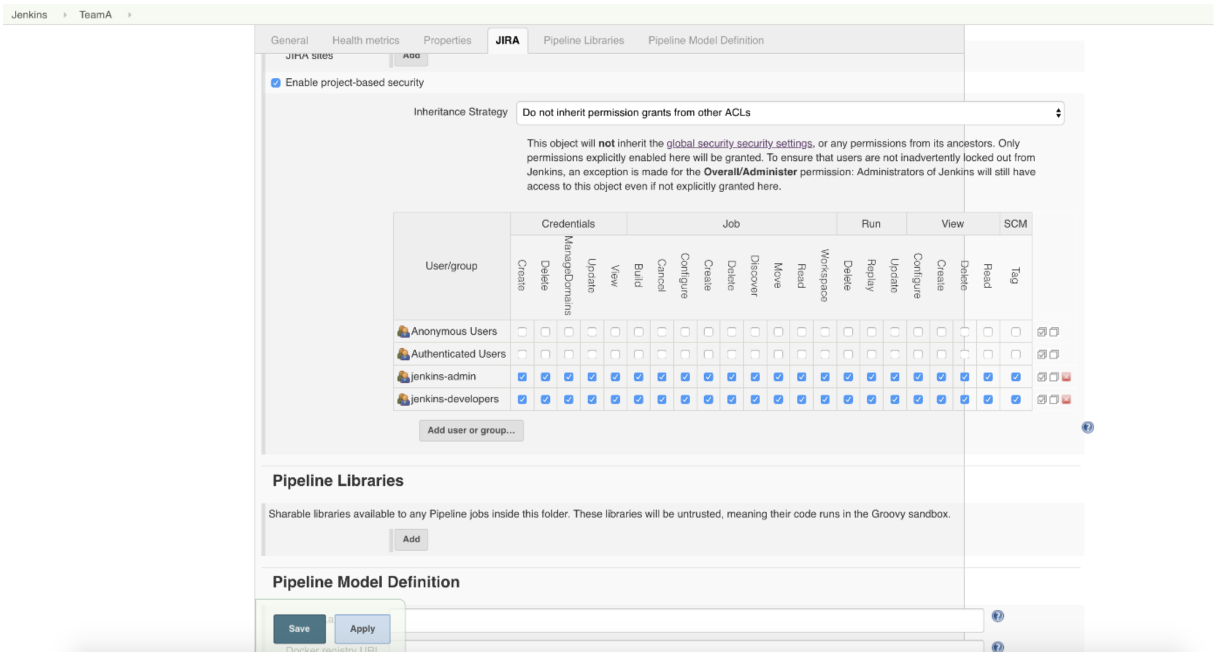
Sadly, the staging environment is shared. We assume that project managers of TeamA and TeamB are well synchronized.

Creating a new folder (TeamA)

First, we will create a new folder called **TeamA**. To do this, sign on as the user `butler` and do the following:

- Click on **New Item** in the left frame
  - **Name:** `TeamA`
  - Kind: **Folder**
- Click **Save**

Review the folder settings on the configuration screen that is displayed and enable Project security to only allow access to members of the `jenkins-admin` and `jenkins-developers` groups.



☑ Enable project-based security

Inheritance Strategy: Do not inherit permission grants from other ACLs

This object will **not** inherit the global security security settings, or any permissions from its ancestors. Only permissions explicitly enabled here will be granted. To ensure that users are not inadvertently locked out from Jenkins, an exception is made for the **Overall/Administer** permission: Administrators of Jenkins will still have access to this object even if not explicitly granted here.

User/group	Credentials					Job									Run			View				SCM
	Create	Delete	ManageDomains	Update	View	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag
Anonymous Users	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Authenticated Users	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
jenkins-admin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
jenkins-developers	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Add user or group...

Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

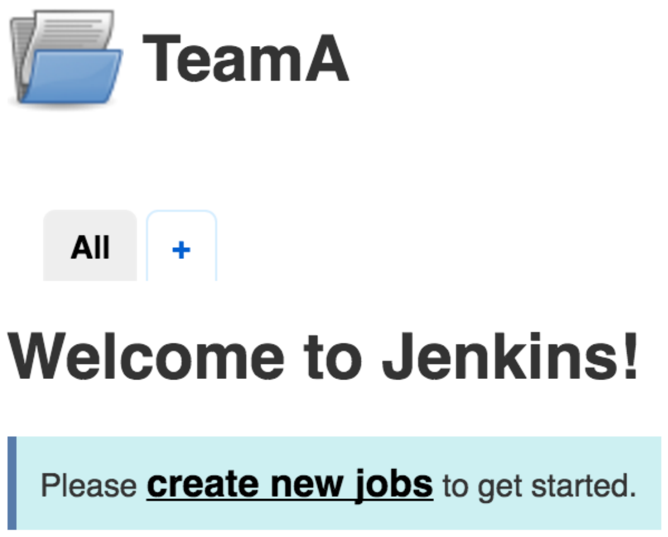
Add

Pipeline Model Definition

Figure 1. New folder for Team A

Add a build to the TeamA folder

You can see that the newly created folder is empty.



TeamA

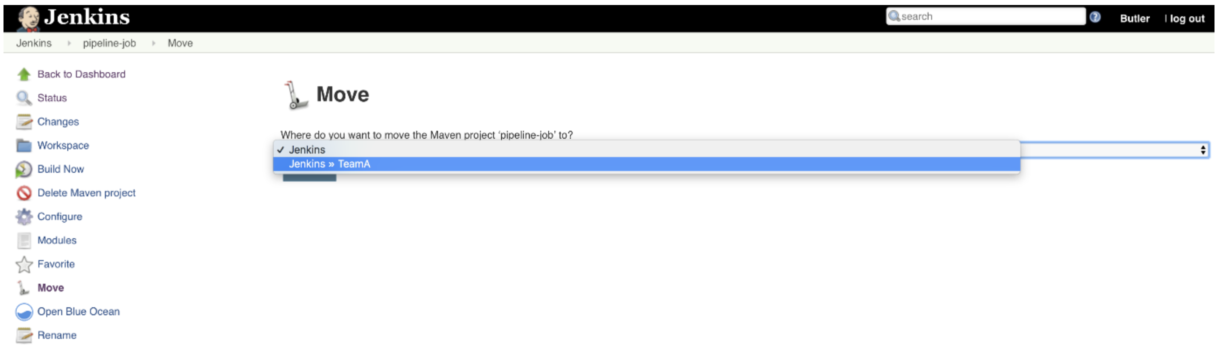
Welcome to Jenkins!

Please **create new jobs** to get started.

Figure 2. Team A folder is empty

To add a Pipeline to this folder:

- Browse to the main Dashboard (use the **Up** button in the left frame)
- For **pipeline-job**:
  - Go the the Job page.
  - Use the **Move** button (in the left frame).
  - Move to the **TeamA** folder.



Where do you want to move the Maven project 'pipeline-job' to?

✓ Jenkins
Jenkins » TeamA

Figure 3. Move Pipeline to the Team A folder

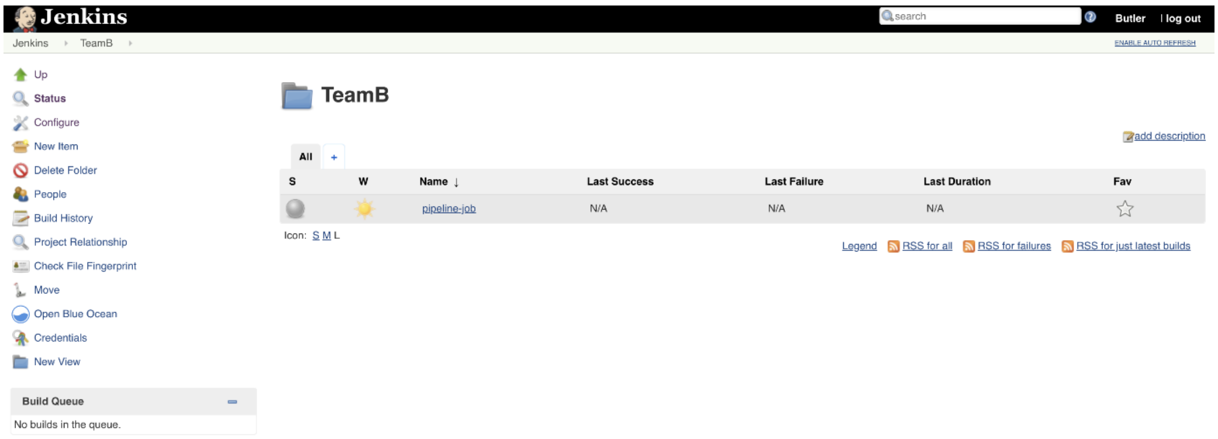
Copying folder to create TeamB folder

We now want to create the TeamB folder using the same settings and content as the TeamA folder. To do this:

- Create a **New Item**:
  - **Name:** `TeamB`
  - Type: **Copy existing Item**
  - **Copy from**: `TeamA`

Review the configuration that is displayed; it is the same as the TeamA configuration. You now have a `TeamB` folder with a **copy** of the original content. Jobs have no **Build** button (yet) and their status is *Grey*

Review the Job configurations *before* you try to launch a build from them.



S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
●	☀	pipeline-job	N/A	N/A	N/A	☆

No builds in the queue.

Figure 4. TeamB folder with newly created jobs

Notice that the pipeline-job in the TeamB folder is greyed out. This is because any job you copy is disabled, or rather, not buildable by default. You usually do not notice this because the form opens and you make modifications and, when you save the job, it becomes buildable. You can disable the project and then re-enable it to see the **Build Now** option on the left hand side menu.

When you are satisfied with the configuration, click **Save**.

Now kick off builds of the `pipeline-job` for **both** folders. See that they behave as independent jobs.

Folders on the filesystem

Folders provide namespacing for Jenkins objects. As you have seen, jobs in different folders may have the same names but are completely separate within their own namespace.

You can see this if you look at the arrangement of files in the filesystem. To do this, go to the command line and execute the following sequence of commands:

1. Spawn a shell on the Jenkins controller:

   ```
   $ docker exec -ti jenkins /bin/bash
   ```

2. List the contents of the `/var/jenkins_home/jobs` directory:

3.
   ```
   $ cd /var/jenkins_home/jobs/
   $ ls -l
   ```

   You should see a directory for `TeamA` and another directory for `TeamB`.

4. List the contents of each subdirectory:

5.
   ```
   $ ls -l TeamA
   $ ls -l TeamB
   ```

   You see that each directory contains a `config.xml` file and a `jobs` subdirectory.

6. List the contents of the `jobs` subdirectory for each Team:

7.
   ```
   $ ls -l TeamA/jobs/
   $ ls -l TeamB/jobs/
   ```

   The workspace for each job is stored here so you see a subdirectory for our `pipeline-job` job for each Team.

8. Compare the `config.xml` files for `TeamA` and `TeamB` and see that they are the same:

   ```
   $ diff -u TeamA/config.xml TeamB/config.xml
   ```

   `TeamB` is a clone of `TeamA`. We have not modified the configuration, so the two files are identical.

That's all for this exercise!