ALM Octane – DevOps

Hands-on Exercise Guide

Designed with NimbusServer-2019.R1 and NimbusClient-2020.R1

Contents

Welcome!

These exercises will guide you through some advanced ALM Octane integrations for the ADM portfolio as used in the Nimbus Demo environment. This information and the associated exercises are for beginner to intermediate individuals who are learning Octane but who also want to know more about DevOps and Octane's various integrations.

Octane is an advanced software planning and tracking tool that integrates with other DevOps tools to provide insight into the complete DevOps lifecycle. You may think that Octane is just an agile management tool but when combined with CI and DevOps tools it provides unparalleled visibility and drilldown capabilities into a wealth of application development details.

Roles

The exercises contained in this guide are specifically built to manage and automate packaging, compiling, deploying and testing software. These tools and techniques are primarily used by integration or automation engineers. The following list shows some other roles that might be involved:

- Integration Engineer
- Automation Engineer
- Developer
- Tester
- QA Lead
- Project Manager

Activities

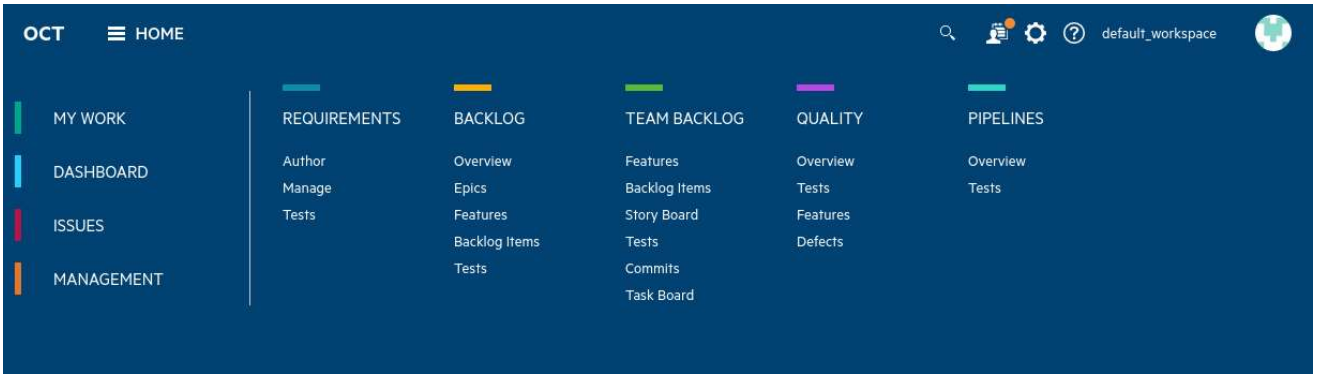
The list below shows the progression of learning activities for this guide:

- Overview and Introduction of ALM Octane
- Building an Agile Backlog
- Jenkins and DevOps Integrations
- Tracking Code with Octane
- Working with Gherkin
- Running the AOS Web Pipeline
- Integrating UFT Tests in Octane
- Integrating Security Testing
- Synchronizing MF Connect with Jira, Octane and ALM
- Integrating SonarQube Code Coverage
- Building the Account Service Module

Let's get started!

Exercise 1: Overview of ALM Octane

ALM Octane Modules

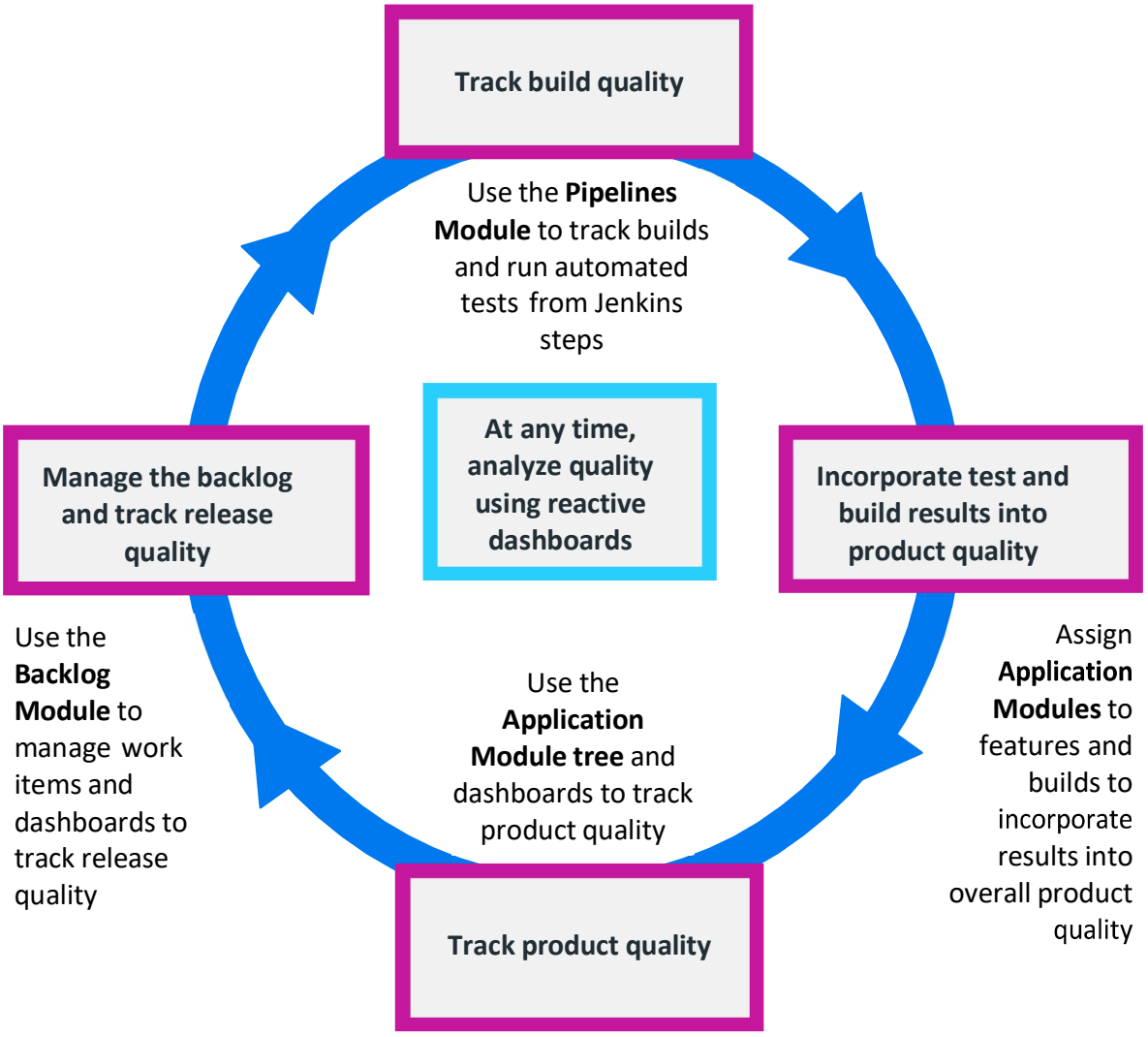


Requirements	The Requirements module provides you with a free-form method of describing what you want to deliver, in terms of ideas and general specifications. Create requirements documents describing what you want to deliver at a high level. You can also link requirements to tests.
Backlog	The backlog is a list of the items you are handling during product development. For each release you define a backlog, which includes epics, features, user stories, quality stories, defects, and their tests.
Team Backlog	Manage the team backlog by assigning items to teams, designating specific team members to perform the work, and track progress of the team. After setting up the overall release plan, work with your team's backlog items to start carrying out your development plan.
Quality	Analyze and track the quality of the product with end-to-end testing, cross-feature functionality testing, and regression tests.
Pipelines	Track the status of builds and test runs included in pipeline runs, and analyze build failures.
My Work	Quickly view the work assigned or related to you.

 **Dashboard**	Use the ALM Octane charts and graphs to analyze quality in context.
 **Issues**	Report and fix defects when running manual tests or analyzing automated test failures. You can also view and manage security vulnerabilities.
 **Management**	Allows you to manage Releases, Timelines, Teams and Team Members without having to enter the Settings section.

ALM Octane Processes

ALM Octane covers different aspects of quality management, including managing backlog items, creating and executing tests, as well as tracking release, build and overall product quality.



For each release, ALM Octane organizes **Epics**, **Features, User Stories** and **Tasks** into a structure called the **Backlog Tree**. Upon completion, QA engineers can design tests in context of the work they are doing. They can create manual tests or BDD Gherkin tests to cover and track quality for a release, including functional, sanity, and acceptance tests. Created tests can then be executed, and if they fail, a Defect can be created, assigned, worked on, and fixed.

In addition, tests can be assigned to specific releases for tracking Release Quality as well as to **Application Modules** for tracking overall Product Quality**.** Quality reports for a product can be monitored by using the provided Dashboard feature. ALM Octane can be integrated with a CI server like Jenkins to future enhance Product and Release Quality reporting and manage a DevOps team.

The Requirements Module

NOTE – In order to start using Octane and view these modules run the following command:

`$ nimbusapp octane:15.0.46.68 start` (Username: sa@nga Password: Password1)

Then navigate to the **Requirements** module and look around.

Requirements can be high-level descriptions or formal documentation depending on your methodology.

You can link requirements to tests, providing you with coverage on each requirement. If your team works with both requirements and backlogs, the two can be linked to one another, showing which backlog item implements which requirement.

For example, you might have a requirement to send astronauts to Mars and bring them back to Earth, and your backlog will be comprised of a huge number of detailed features, user stories, and tasks.

You can also use requirements to document business-oriented information rather than simply defining deliverables. For example, within requirements you might enter business objectives, executive briefs, risk factors, or market opportunities.

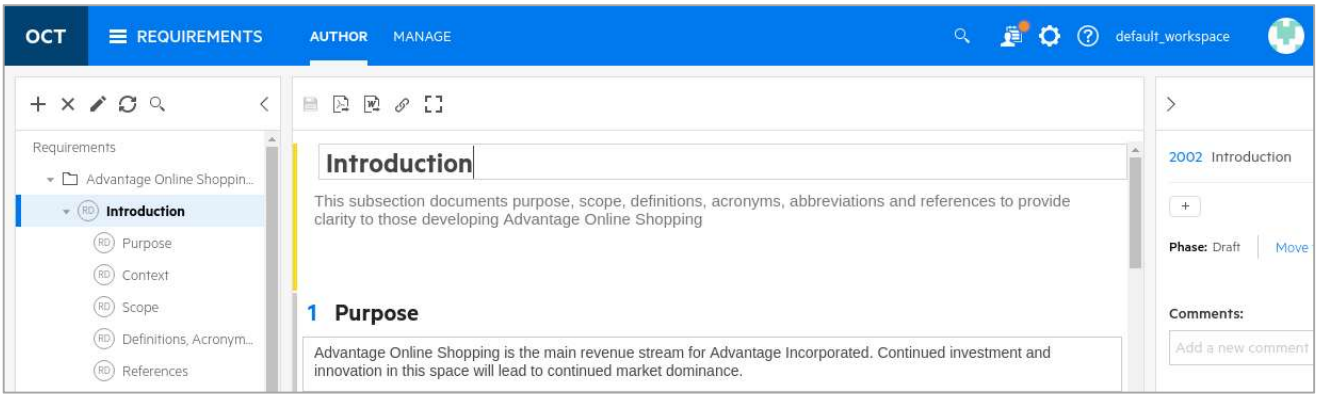
How are requirements related to backlog items?

From a use-case perspective, the Backlog module is project-management oriented. Backlog items are defined in a hierarchy of epics, features, and stories, and different backlog items are created for different teams (for example back-end vs. client).

Requirements contain the overall story of what you want to deliver, rather than a hierarchy of tasks. For example, you might define a requirement as "Multi-language support," and this could be mapped to ten different items in the backlog.

This is also reflected in the personas using these modules; backlog items may be written by project owners or PMOs, while requirements may be written by business analysts or PMs.

From the perspective of test coverage, you could work with both backlog and requirements in parallel. Alternatively, you could choose one of these modules to reflect quality implementation, and link your tests to that module.



The Backlog and Defects Modules

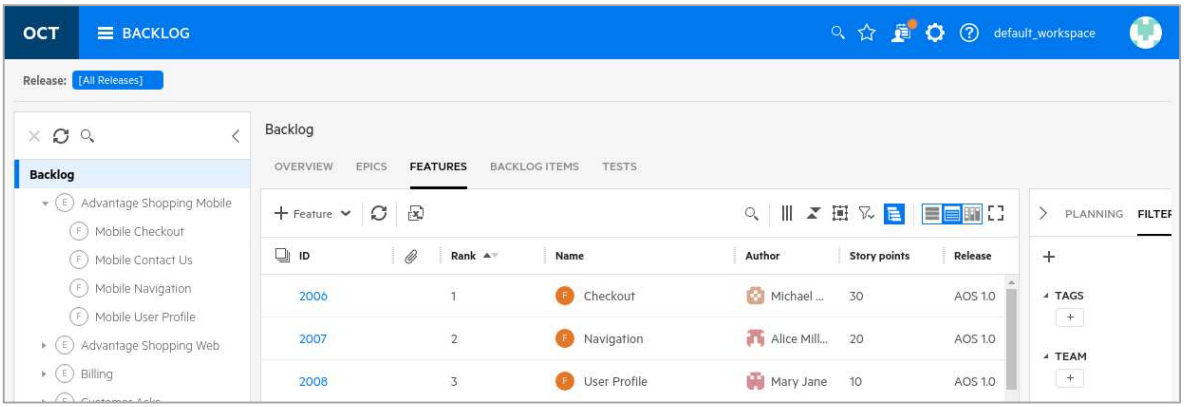
The backlog is simply a list of work items identified during product development. Defects are a special class of backlog items that identify problems that need correcting. In ALM Octane, the backlog is managed using the Backlog module. The Backlog module contains all the product's epics, features, user stories, quality stories, defects, and their tests and is probably where most users do the majority of their work.

The Backlog module contains:

- The Backlog tree containing the hierarchy of epics and features, regardless of release
- A grid containing separate tabs for epics, features, backlog items (user stories, quality stories, and defects), and tests. The items displayed in the grid are dependent on the item selected in the tree, and selected releases and sprints, or any additional filters

The Backlog module enables you to:

- Create epics and features, user stories, or quality stories
- Open defects for problems encountered during development, testing, or deployment
- Rank items and plan development cycles
- Add tests to epics, features, user and quality stories, or defects
- Track release and sprint development progress

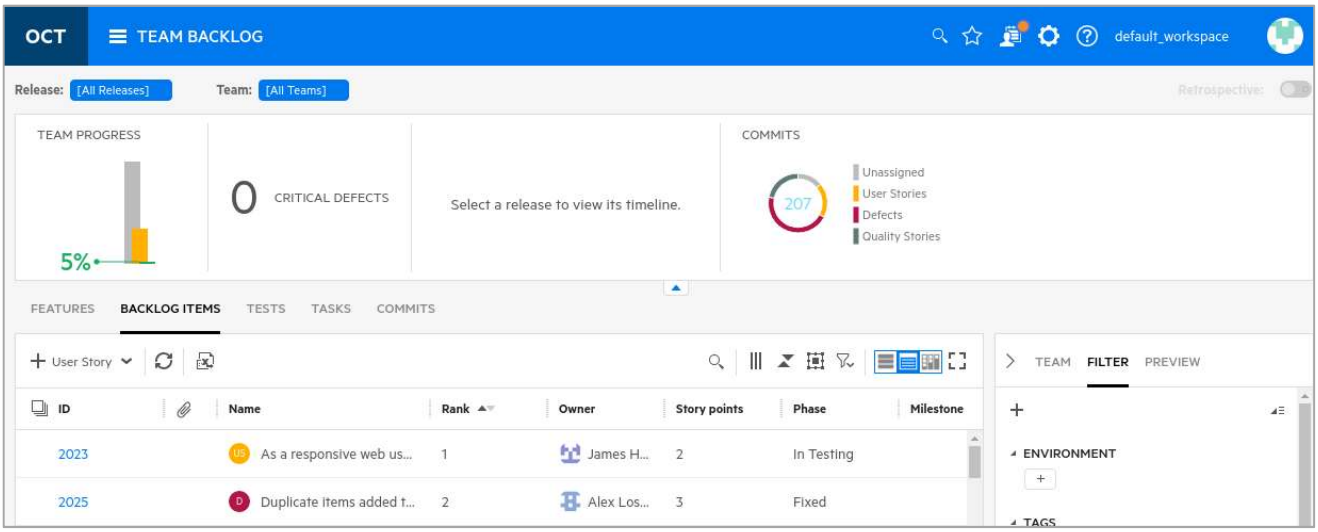


The Team Backlog Module

Teams and cross functional collaboration are an integral part of agile development. After setting up the overall release plan, you work with your team's assigned items to start carrying out your development plan. Do this from the Team Backlog module. You can edit your team's capacity per sprint, assign items to team members, track team progress, create user stories and tasks, and create and execute tests.

The Team Backlog lets you manage your team's work by focusing on your team's sprint activities and provides views tailored to make it easy to track your team's status. You can even view code commits related to a specific release.

You can also easily filter your team's backlog items and plan your team's capacity.



The Quality Module

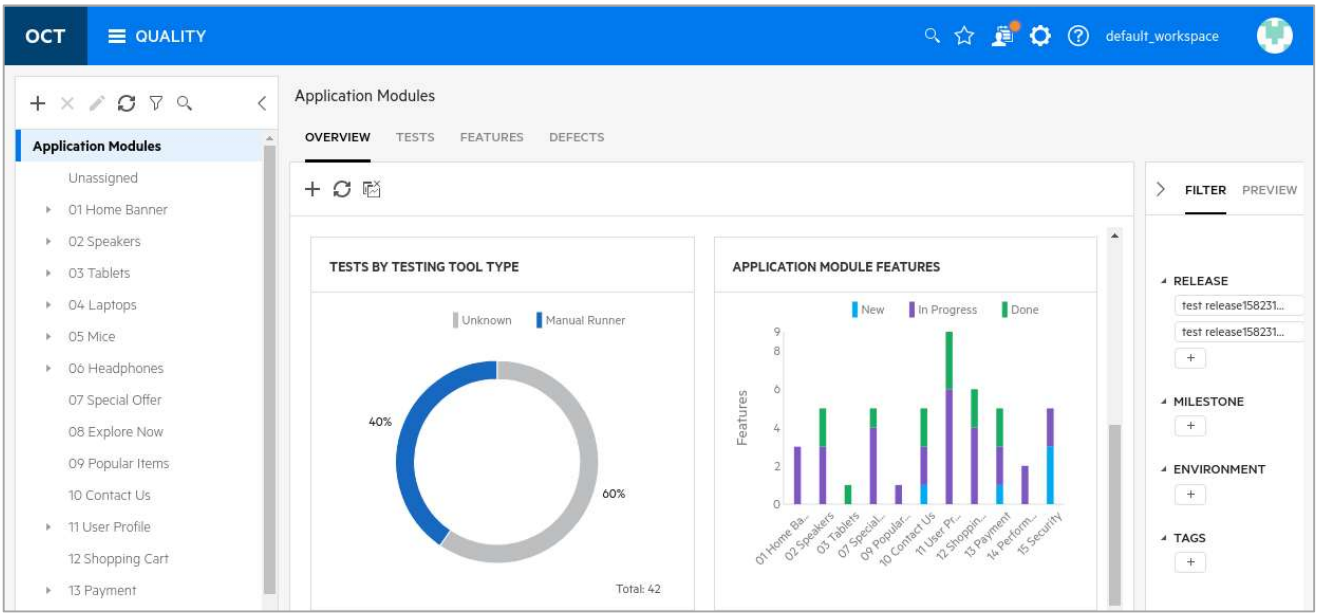
In ALM Octane, you have the ability not only to monitor the progress of your work but also to track product quality across all areas of your application and project. This is unique to ALM Octane since it measures quality in a number of different ways, including test results, defects, or feature quality status. You can view this quality in the Quality module and also through the various dashboards.

Application modules are unique to ALM Octane and represent a functional breakdown of the product. These application modules are linked to features, defects and tests. It's through these tests that the overall application quality is determined.

Using application modules is important, as in many cases it is seen that many of the reported defects and testing problems that occur are limited to a small number of areas within an application. By setting up application modules, and associating tests, defects, and features, it helps you see the problematic areas of the application and concentrate testing efforts and development effort toward these areas.

Application modules are release-agnostic, which means that the details you view about the application module are global across all releases and reflect the overall health of the product.

In the Quality module, application modules are displayed in a tree. This tree is a visual representation of your application, organized hierarchically.



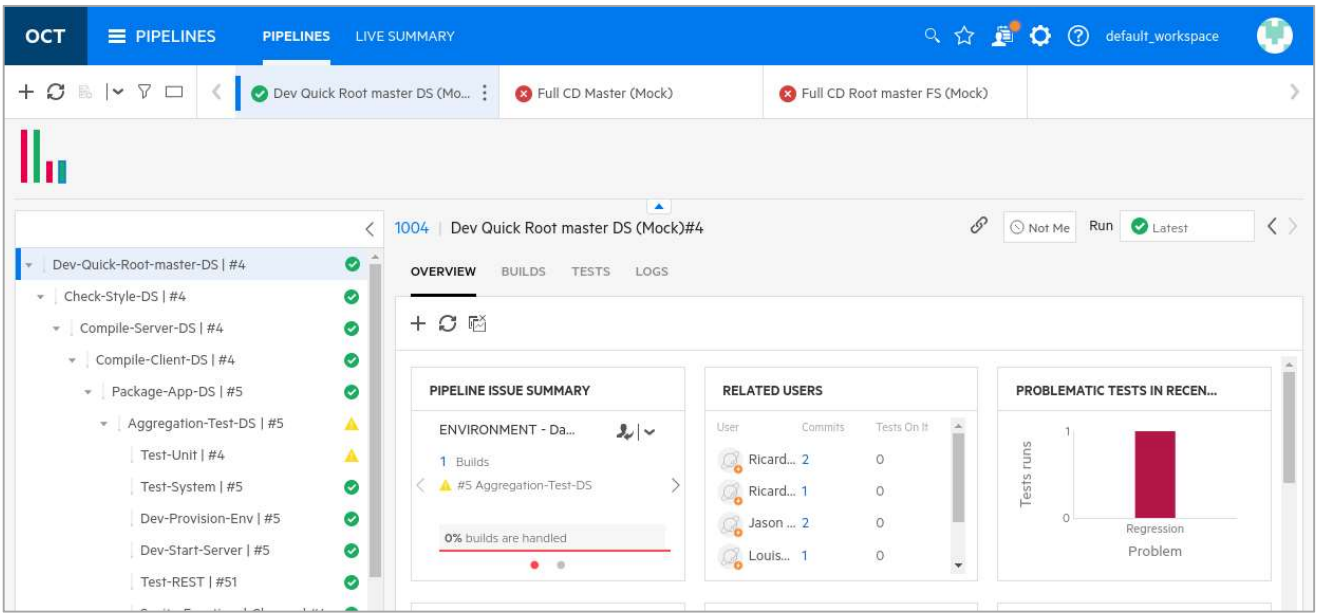
The Pipelines Module

Pipelines in ALM Octane represent the jobs or steps that run on your CI server. ALM Octane incorporates data from your pipelines into your application delivery process, helping you analyze quality, progress, change impact, and more.

After you set up ALM Octane to integrate with your continuous integration (CI) server, you can create pipelines in ALM Octane.

Pipelines represent the flow of your CI server jobs. If you are working with Jenkins or Bamboo, the graphical representation of the pipeline also shows the hierarchy of the jobs in the flow.

When you run a pipeline after adding it to ALM Octane, ALM Octane collects build, test run, and commit information from the pipeline run.



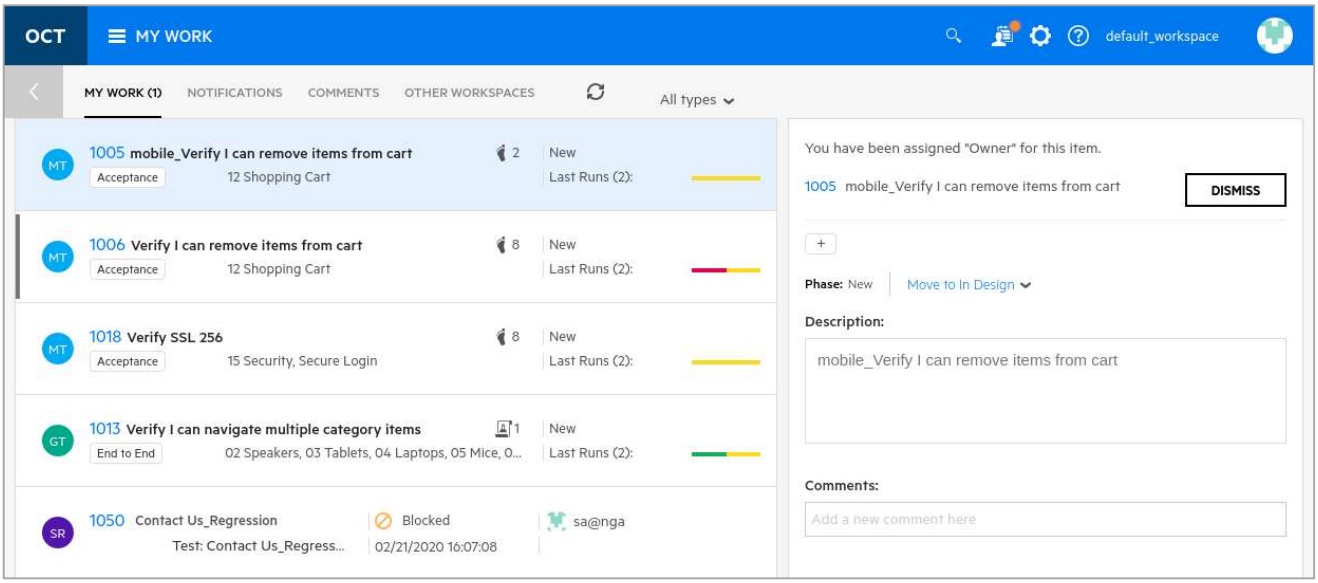
The My Work Module

Track your work with a personalized to-do list of work items. This enables you to quickly view and complete the work assigned or related to you.

By default, items are displayed in the My Work page of the owner of the item. It is also possible to display an item in the My Work module of other users. To do this, your workspace admin creates a rule to assign the item to these users.

The following items are listed on the My Work page:

- Assigned items (Requirements, Backlog Items, Tests), assigned either via a business rule or added manually by another user
- Test runs
- Mentions in comments
- Notifications on followed items



The Dashboard Module

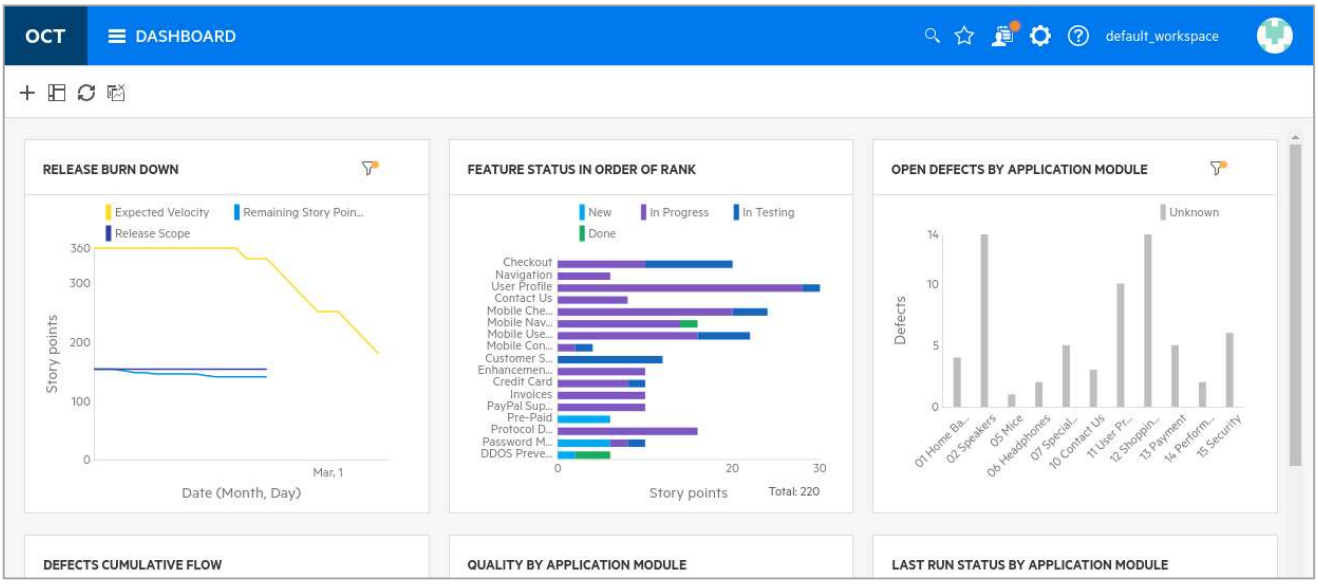
The ALM Octane Dashboard is the control center for analysis of your application's development and quality. It gives a visual, customizable display of how the application's development is progressing and the level of quality.

The dashboard is a collection of widgets (graphs and charts) designed to share data about your product development and progress. Out-of-the-box widgets include widgets to measure product quality within and across releases, progress within a release, test results, and other analytics. If the out-of-the-box widgets are not sufficient for your analysis needs, you can create a custom widget to display your data.

NOTE: Charts and graphs are available in the Backlog, Team Backlog and Quality modules (called Overview) as well as in the Dashboard. The difference is that those should be designed to cater to their specific module while the Dashboard module is an overall collections of metrics and health.

For the dashboard widgets, it is possible to customize the information shown in the dashboard such as the type of data reported, the scope and time frame, and how the data is grouped and displayed.

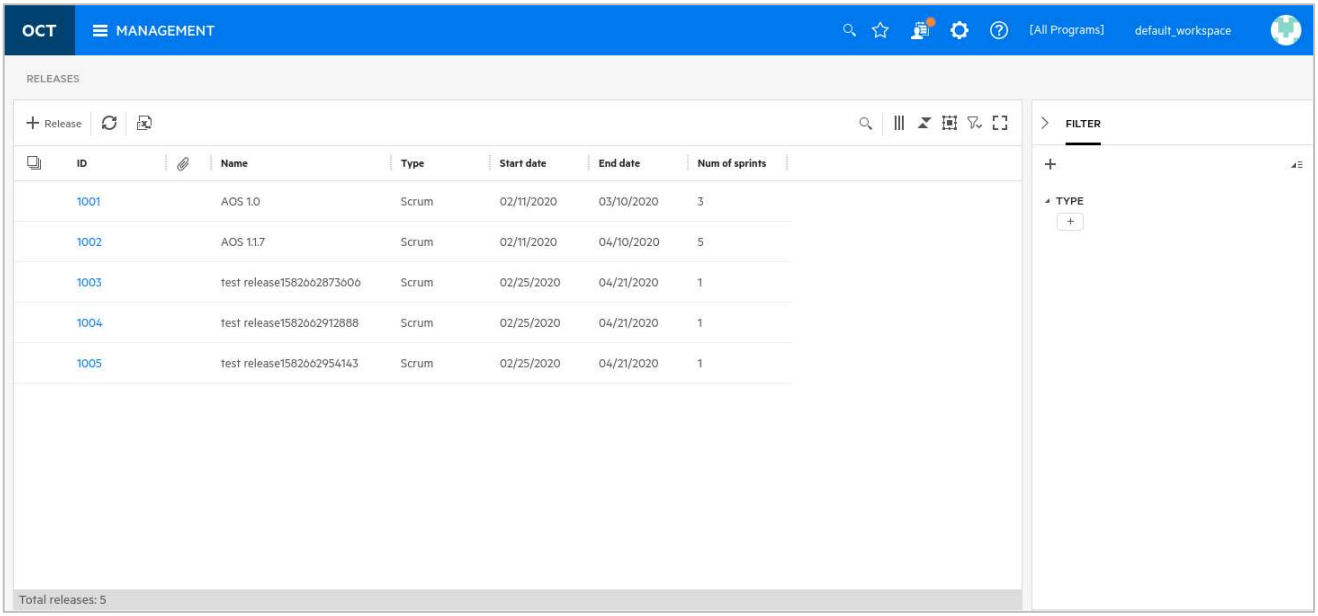
The dashboard also enables you to configure how you want to view widgets and save different widgets as favorites to use on repeated visits to the dashboard.



The Management Module

The Management module currently serves to allow select individuals to edit the releases. Releases can be created and edited in this section and can be changed between Scrum and Kanban along with the start and end dates along with the number of sprints.

Additional basic management functionality may come into this module in the future.



Exercise 2: Introduction to Octane

ALM Octane forms the nerve center of the ADM portfolio. From Octane you can design and manage your application, backlogs and sprints and even run pipelines from Jenkins (via the devops container). Next we'll start the devops, octane and SSC containers so we'll have a CI system, an agile lifecycle management system and a software security system at our fingertips.

Starting the DevOps, Octane and SSC Containers

1. If you haven't already, open a **terminal** on **NimbusServer** and start the DevOps container:

   ```
   $ nimbusapp devops:2.2.1 start
   ```

2. If Octane and SSC aren't running, start them from the **terminal** on **NimbusServer** by typing:

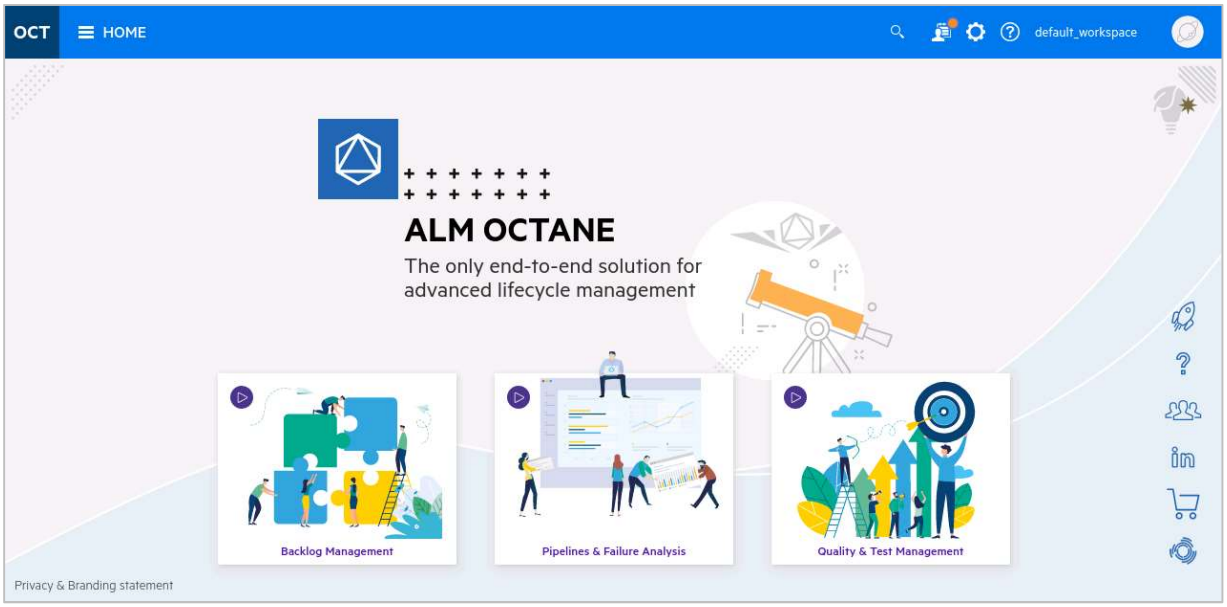
   ```
   $ nimbusapp octane:15.0.46.68  start
   $ nimbusapp ssc:19.1.0 start
   ```

3. Verify that your containers have started properly by typing:

   ```
   $ docker ps      (or dps)
   ```

4. These three containers take a little while to start up so **monitor** your Linux CPU activity with the desktop icon **System Monitor**. Once the activity settles down to under 30 % CPU, these applications are probably ready to interact.
5. Once the activity has settled down, open a browser tab to the **Octane** shortcut and log in.

   Username: **sa@nga**
   Password: **Password1**



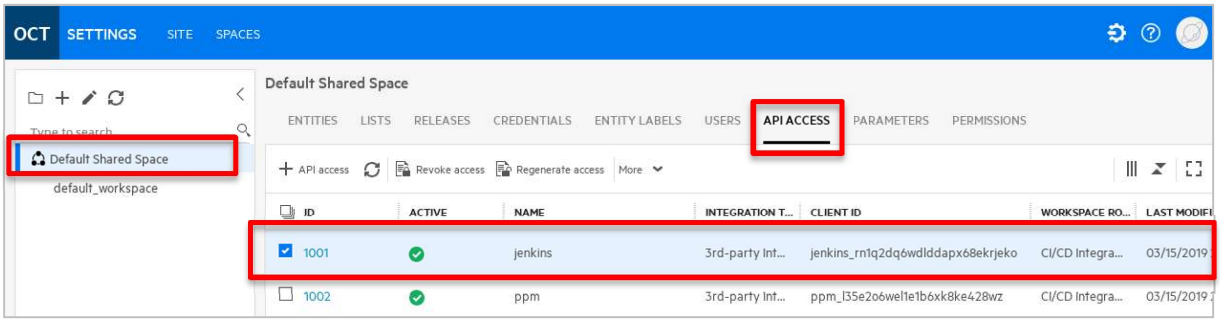
Configuring the Octane and CI Server

Octane and Jenkins (housed in the devops container) need to communicate. An Octane administrator sets this up by creating an **API Access** key. This has already been done in the Octane container and these values were placed into the devops container inside a Jenkins setting.

However, as Octane versions progress, and the Octane plug-in for Jenkins gets updated, these values can get out of sync. Fortunately, we can check this connectivity and correct it if needed.

1. From Octane select the **Settings** cog and navigate to the **Spaces** section.

2. Select **Default Shared Space** on the left and choose the **API Access** tab.



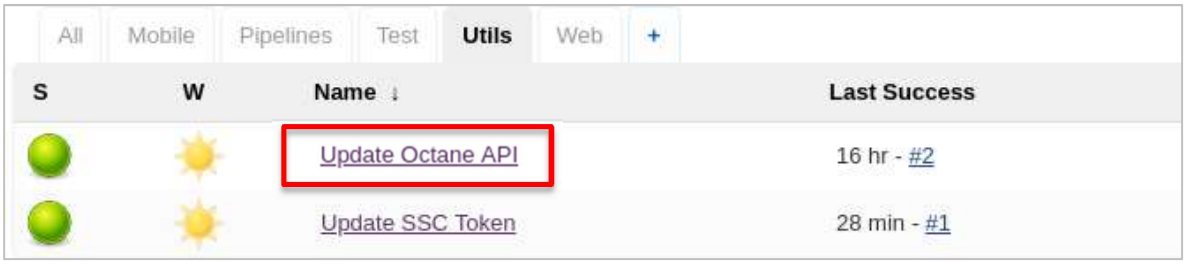
3. Look at the list and find the entry for Jenkins. This line contains the **Client ID** that needs to match with that field in Jenkins. There's also a **Client Secret** that is paired with this value but it's only visible when you first create the key (hence the secret part).

4. Open a tab in your browser to the Jenkins shortcut and click on **Manage Jenkins** on the left menu.
5. Click on the **Gear** icon to **Configure System**.
6. **Scroll** down to the section labeled **ALM Octane CI** and note the **Client ID** – the value should match what you saw in ALM Octane. If it doesn't, that's OK – we'll fix that later.

7. **Click** on the **Test Connection** button. It should display **Connection successful**.
   If it shows a successful connection, go to the Jenkins home page and skip to Exercise 2.

   If it doesn't display this message, then you could copy the value from Octane into this field OR use a special utility (described below) that will also fix this situation.

8. Open a **second browser tab** to **Jenkins** and select the **Utils** tab on the top.
   The **Update_Octane_API** job in Jenkins will automatically update your client secret and key.



9. Select the **Update_Octane_API** job link.

10. Read the description for this job. It mentions that when you run this job, you won't see its progress (it runs very fast) and will restart Jenkins so you'll need to refresh your browser to see if it worked.

11. Click on **Build Now** to run this job and then click to refresh your browser.
    This can take a couple of minutes – please be patient.

12. Go back to your home **Jenkins** tab that is on the configure page and click the **Test Connection** button.
    It should show **Connection successful** this time even if it didn't before.

    BONUS: As a reward for finishing this section, try this command from your terminal:

    ```
    $ telnet towel.blinkenlights.nl
    ```

    To stop this display, close the terminal and open a new one.

Configuring the Software Security Center (SSC) Server

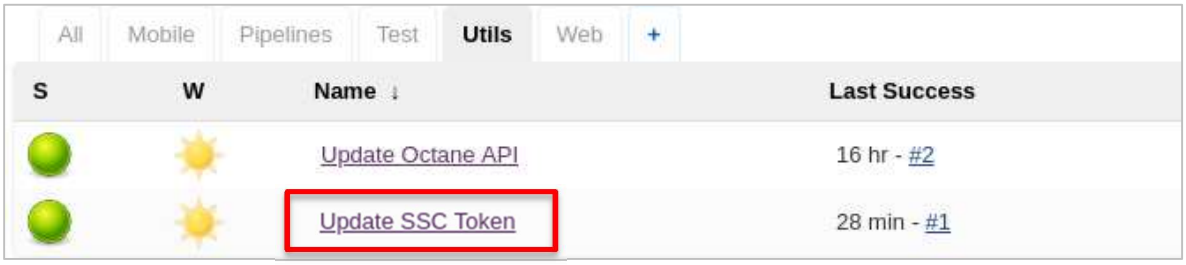
Similar to what we just did by configuring settings in Jenkins for Octane, next we'll configure Fortify Software Security Center (SSC) to work with Jenkins as well. Just like we had a script to do this for Octane, we also have a script to do this for SSC.

1. If SSC isn't running, start it from the **terminal** on **NimbusServer** by typing:

   ```
   $ nimbusapp ssc:19.1.0 start
   ```

   It can take a couple of minutes for the server to finish starting up.

1. Open a **browser tab** to **Jenkins** and select the **Utils** tab on the top.
   The **Update_SSC_Token** job in Jenkins will automatically update your client secret and key.



2. Select the **Update_SSC_Token** job link.

3. Click on **Build Now** option on the left to run this job and then click to refresh your browser.
   This can take a couple of minutes.

Nifty!

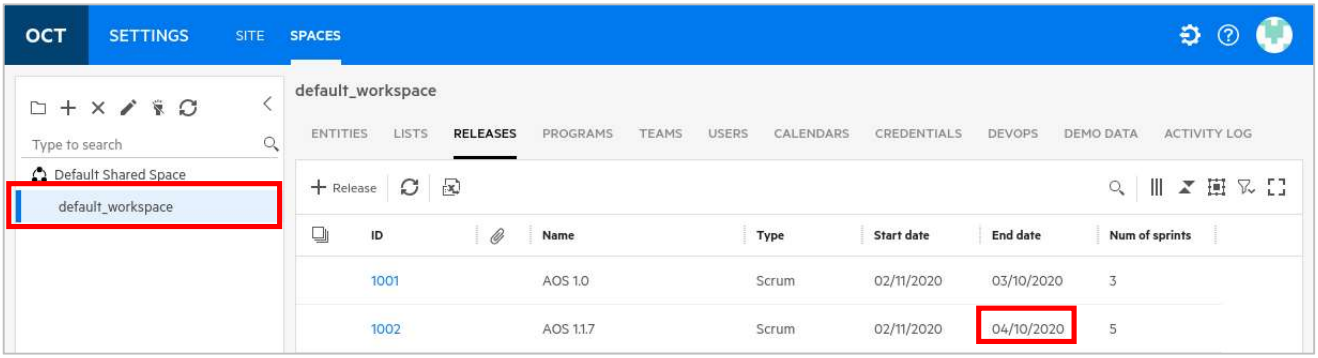
You have just configured the octane, devops and ssc containers!

Exercise 3: Configuring Octane's Workspace

In order for Octane's data to be connected, relevant and current we need to configure a couple of items first.

Setting up Releases

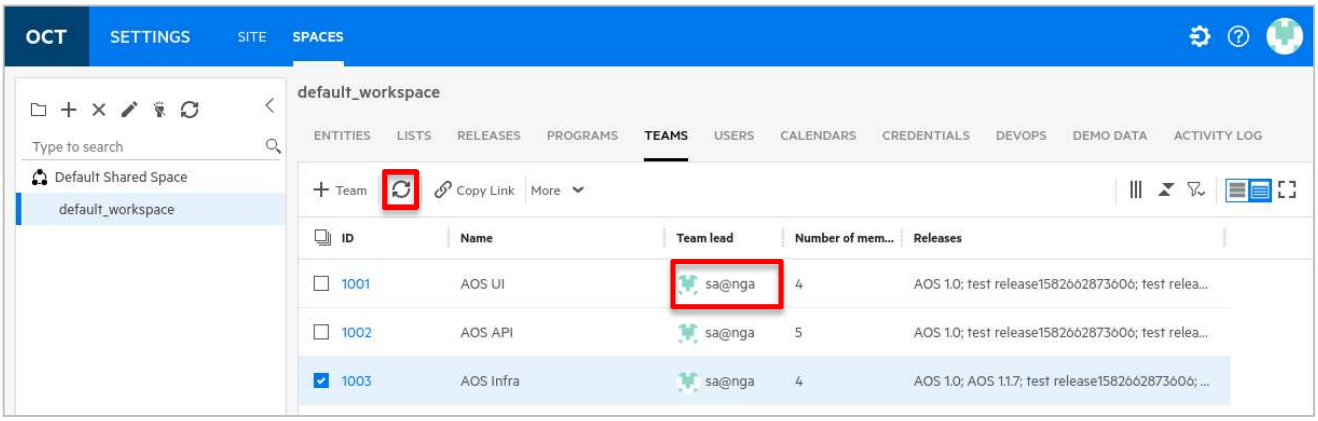
1. Verify that your Octane container is running (`$ docker ps` (`or dps`))
2. Open your browser and **Login to ALM Octane**: http://nimbusserver.aos.com:8085/ui
   a. Username: **sa@nga**
   b. Password: **Password1**
3. Click the **Settings Icon**  and select **Spaces** to get to the Octane administration module.
4. Select the **default_workspace** section on the left and then the **Releases** tab and set the **END DATE** for the **AOS 1.1.7** release to be one month from today's date. That properly adjusts the current sprints.



Setting up Teams

We also need to set up some teams to accurately portray the people in Octane.

1. Select the **TEAMS** tab verify that the **AOS_UI, AOS_API** and **AOS_Infra** teams exist.
2. Select the **MEMBERS** tab and click on the **Team Lead** section and type sa to add **sa@nga** as shown. Do this for all three of the teams (AOS_UI, AOS_API, AOS_Infra).
   Click the refresh option to update the number of members count.



NOTE: You can also do this by clicking the numeric link and selecting the **DETAILS** tab.

3. Click on the numeric link for **AOS Infra** (1003) and select the **MEMBERS** tab.
4. Add a team member by clicking on **Assign Members** pull-down.
    a. Add **Alice Miller** to this team.

Setting up Programs

A program represents an "important, ongoing system development mission" as part of the SAFe initiative.
For more details, check out this link from the SAFe foundation:
https://www.scaledagileframework.com/program-level/

1. Click the back arrow and make sure you have the **default_workspace** selected on the left.
2. Select the **PROGRAMS** tab and add a Program called **Back Office**.
   You can associate backlog items (features, user stories, quality stories, and defects) with programs, and then filter the Backlog module grid accordingly.
3. Click the back arrow and then select the **RELEASES** tab and add the **Back Office** program to **AOS 1.1.7**.
4. Click the gear icon and select **Return to main application**.

Exercise 4: Navigating Octane's Advanced UI

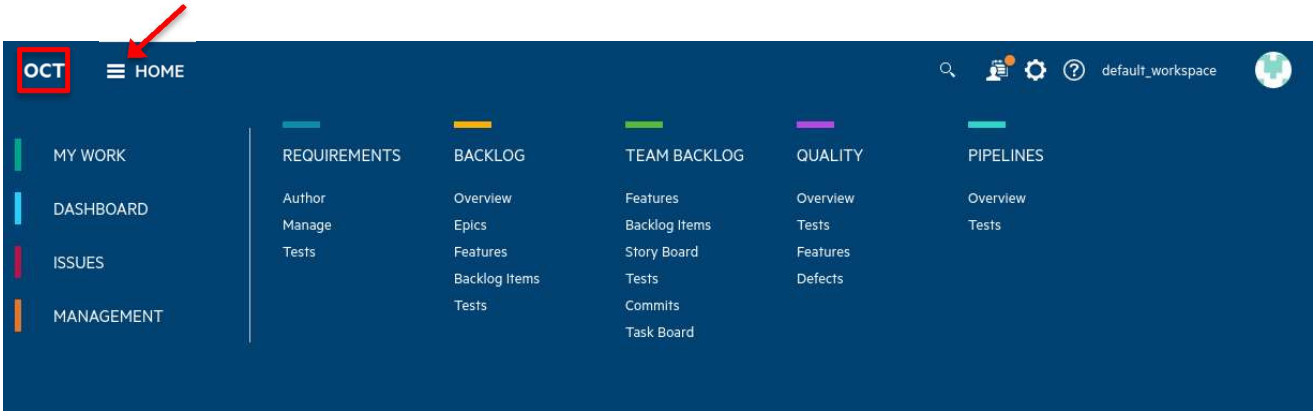
Octane is built on an advanced UI toolkit (Angular JS) that offers simple, easy-to-use controls and an optimized Elastic Search engine for fast UI response. Octane is available on premise, as a docker image or on SaaS.
In fact, an older release of Octane is actually used to develop the next release of Octane!

Octane's architecture actually consists of three parts – the Octane server, the database server and the elastic search server. For purposes of simplicity these have all been combined into a single docker image in the Nimbus demo environment.

Navigating Module Functions

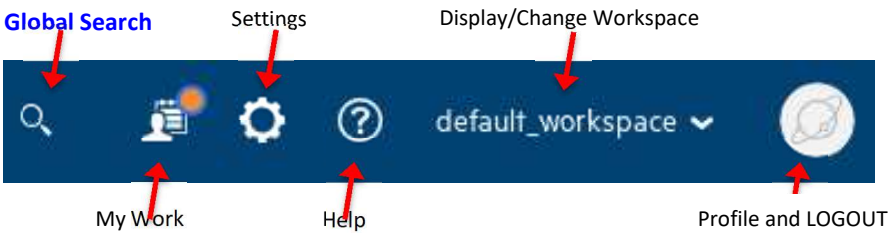
1. **Verify** that the octane container is running.
2. Have your browser open and **login** to ALM Octane (sa@nga, Password1)
3. **Select** the **OCT** home module to get to the HOME starting page.
4. **Click** the triple bar  to bring up the module navigation menu.

Clicking on the OCT brings you HOME



5. This navigation bar is how you'll switch modules easily. Try clicking some of the links and then **click** the triple bar  again to get back to the navigation menu.

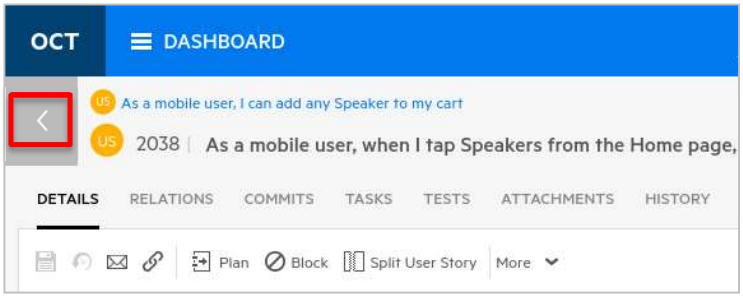
6. At the top right corner are some special icons.



7. **Click** on the Global Search magnifying glass and type in **2062**
   This will bring up the details of a User Story (US) whose ID is 2062 and also displays the item's name:

"As a mobile user, I can add any Speaker to my cart."

8. **Clear** the search entry and type "**Speaker**" instead.
   This will search for user stories that have "Speaker" in their name or description.
   Click the numeric link next to the second item (**2038**).
   This will navigate you to the details of User Story **2038**.

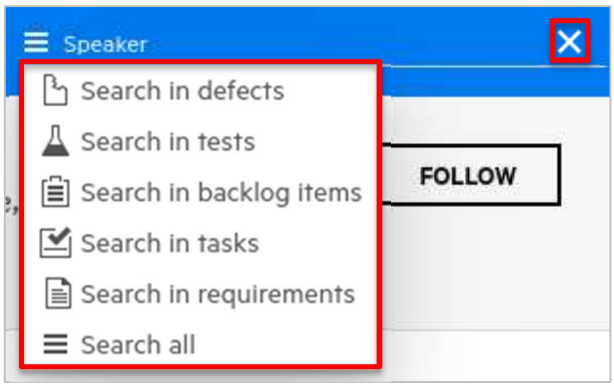


9. **Click** the back arrow as shown above to get back to where you started.
10. Next to the word Speaker in the global search bar is something that looks three horizontal bars.

This icon represents searching in backlog items only.
Click this icon to show the full list of search categories including **Search All**.

NOTE – if more than a couple items appear during a search you may need to click the link to see them. (e.g. Backlog Items(4))

11. **Click** the **X** to clear the Speaker text from the global search line.

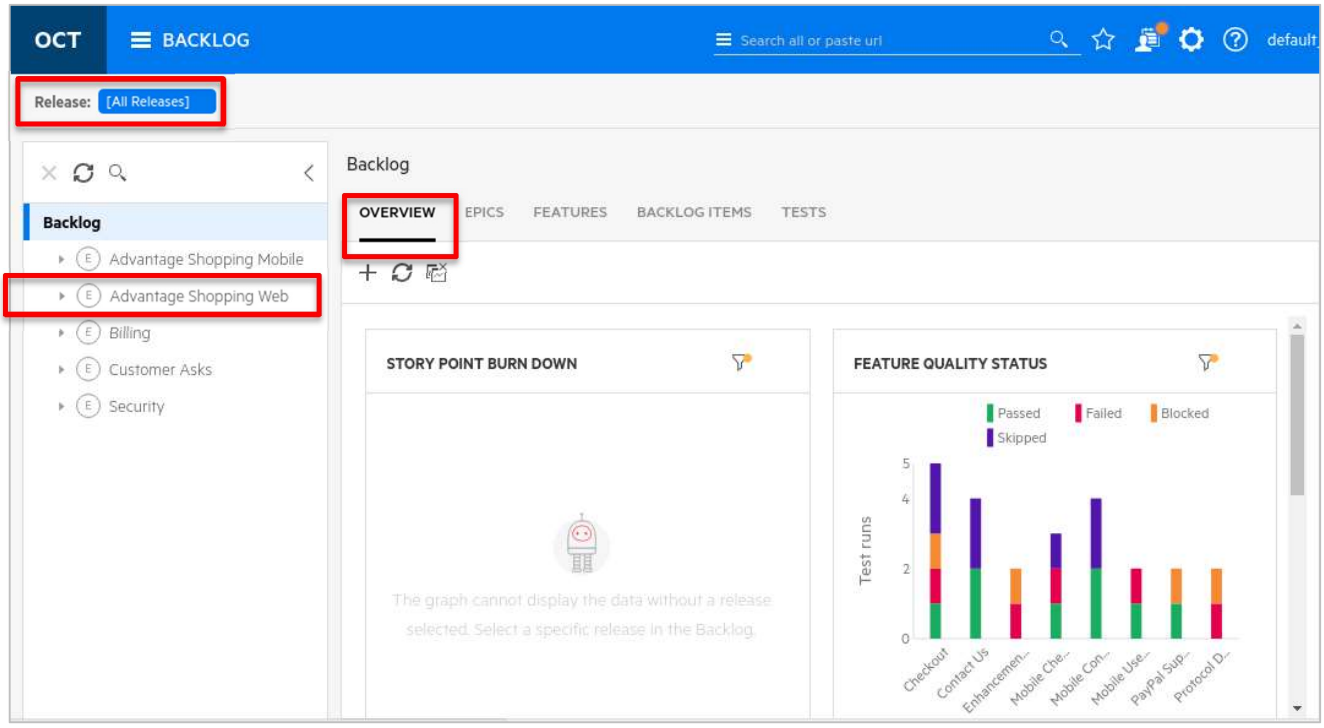


12. **Click** the bars icon again to dismiss this menu.

13. **Click** the OCT triple bar  again and select the **Backlog** icon or text.

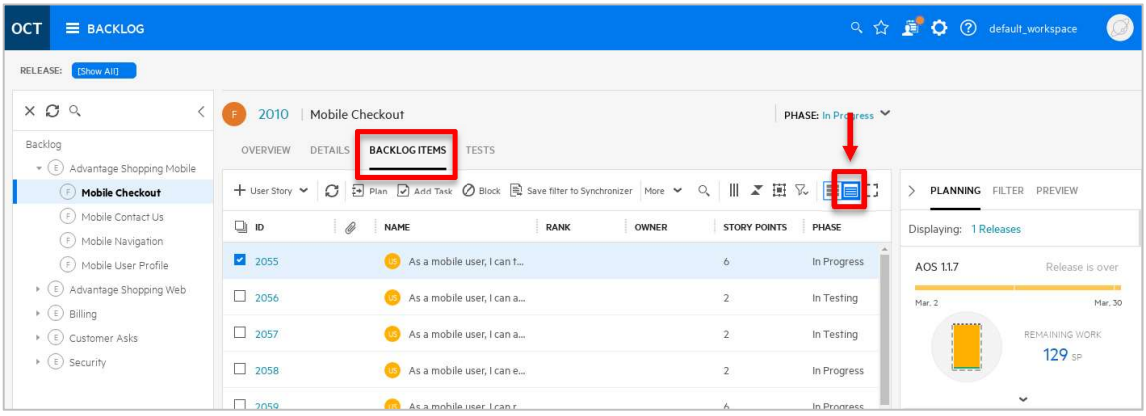
Manipulating Octane's Controls

We're now on the **Backlog** page but not everyone will see the same view. This is because Octane remembers your settings, tabs, filters and many other selections that your account last had.



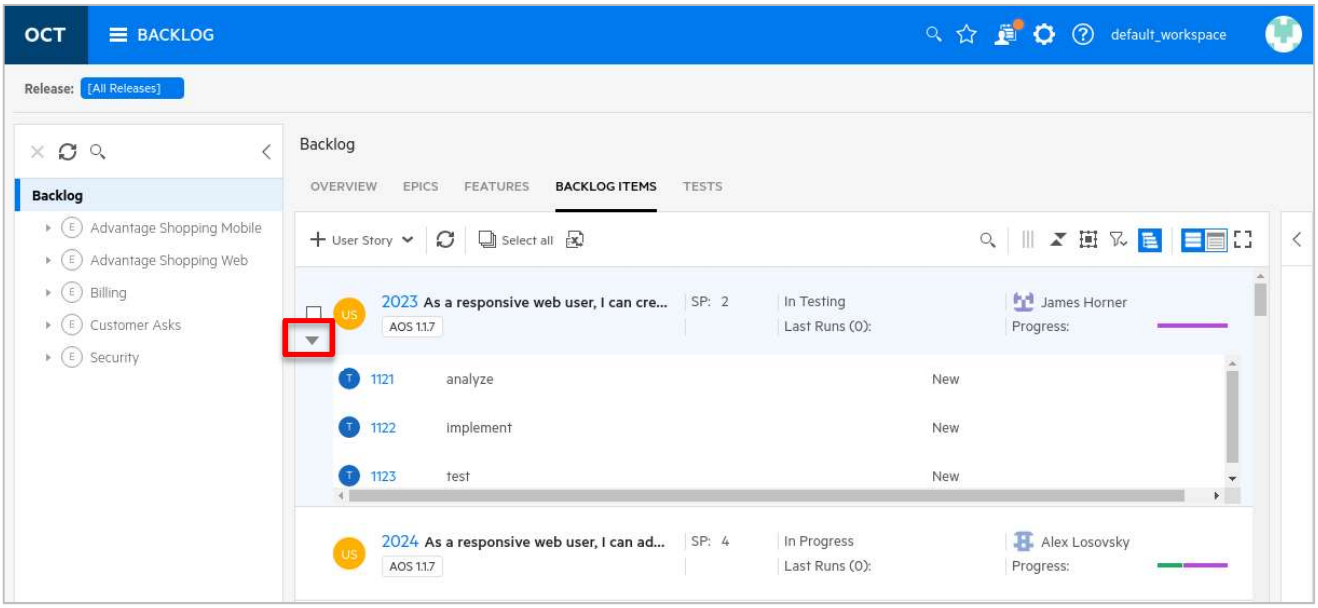
1. **Modify** your settings (**Backlog** tree, **Overview** tab, etc.) so it shows the dashboard view shown above. Most modules offer an Overview mode which is a contextual dashboard for that section. From this section you can create graphs and charts specific to the backlog items.

2. Select the **Backlog Items** tab and the **Grid View** icon.
   This view shows a convenient grid view of all the backlog items. But it gets even better!



3. Select the **Smart List View** to expose more details on each selected user story.

4. From the Smart List View, **select** the topmost user story and click the **expansion triangle**. Now, in addition to a grid-like view you can easily navigate to a user story's tasks.



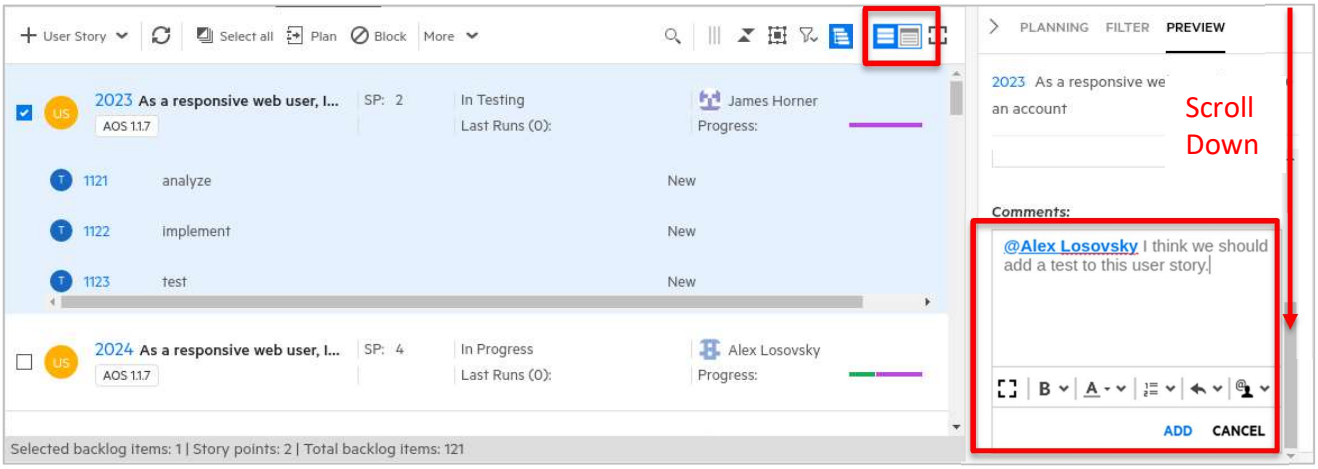
5. Another advantage in this view is the ease of adding Comments. Comments are notes from developers, scrum masters, management, etc. which offer some question or actions for individuals. ALM Octane even recognizes user names if you precede them with an @ symbol.

   Open the **PREVIEW** tab on the right and **type** the following into the **Comments** section (don't use copy and paste):

   When the Alex Losovsky box appears, select it.

   @Alex
   Alex Losovsky

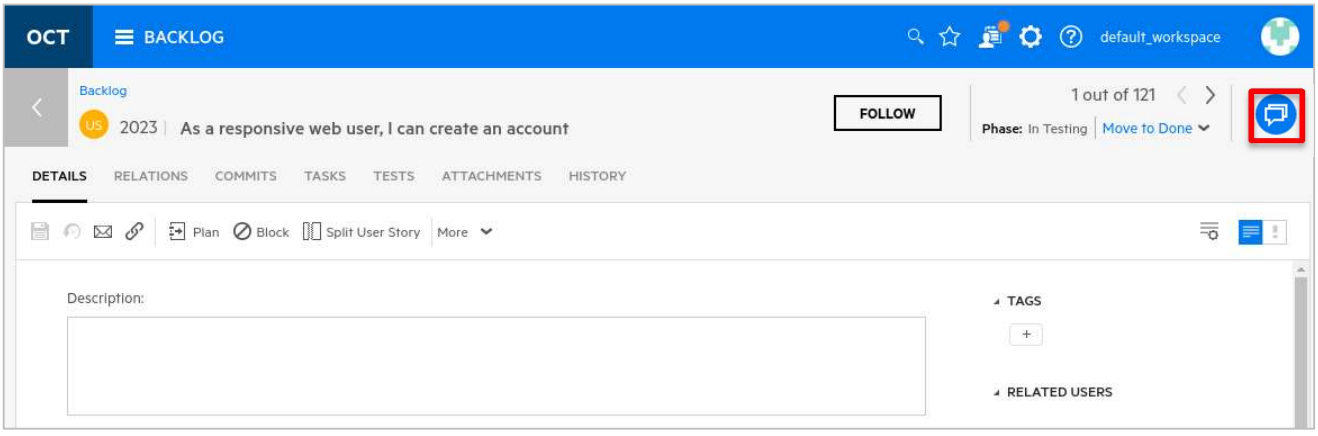
   I think we should add a test to this user story.



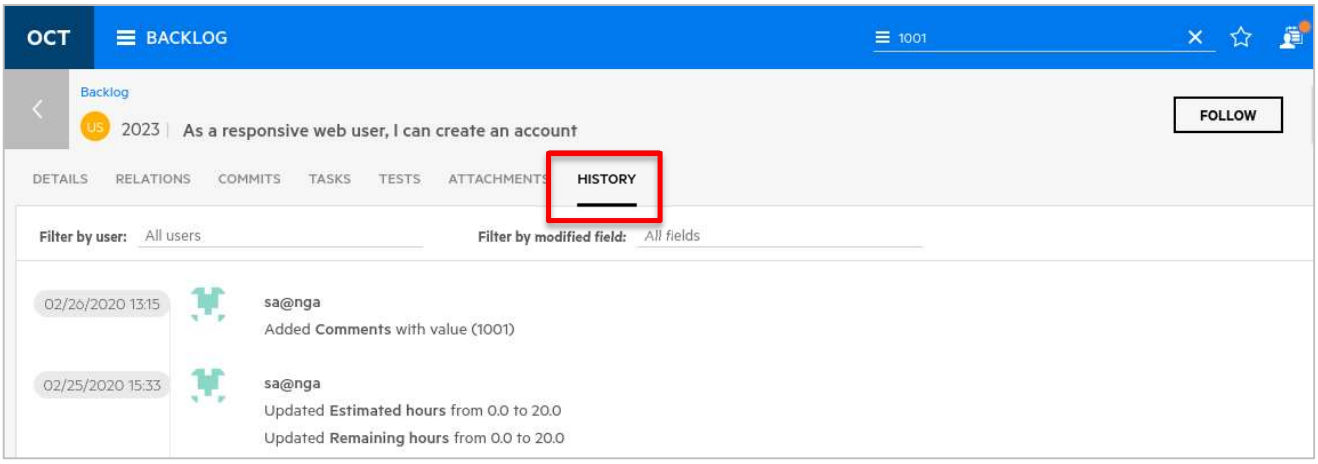
6. Click **ADD**.
7. Click the numeric link for this topmost user story to get to its details page.

8. The details page allows you to edit every field for this User Story (US).
   BUT WAIT – Where is the **Comment** field I just edited?

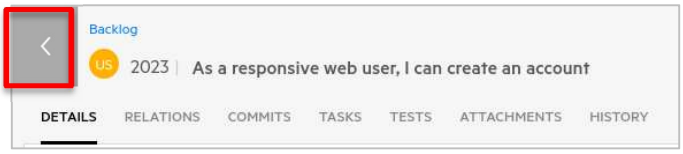
   If you click the circular blue icon in the upper right you can view or add new comments too.
   You can also **pin** the Comments box so it stays up on the right (try it!).



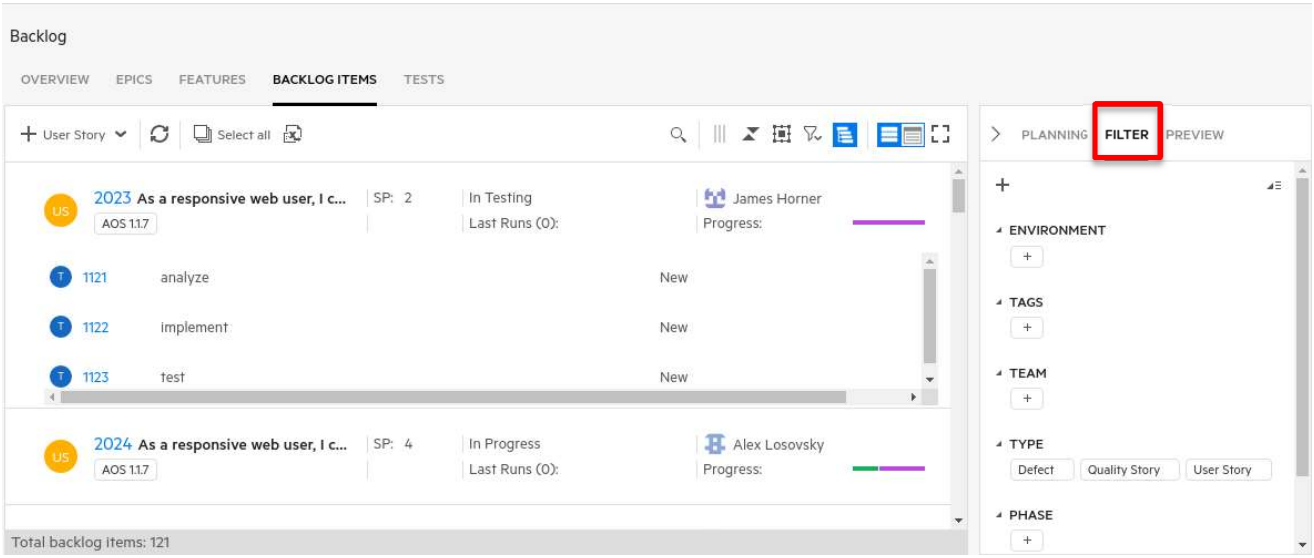
9. Click the History tab to see a history of changes to this User Story. Everything is tracked.
   You should see your change to the **Comments** at the very top of this list.



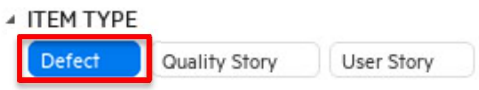
10. **Click** the back arrow symbol to return to the Backlog Smart List View.

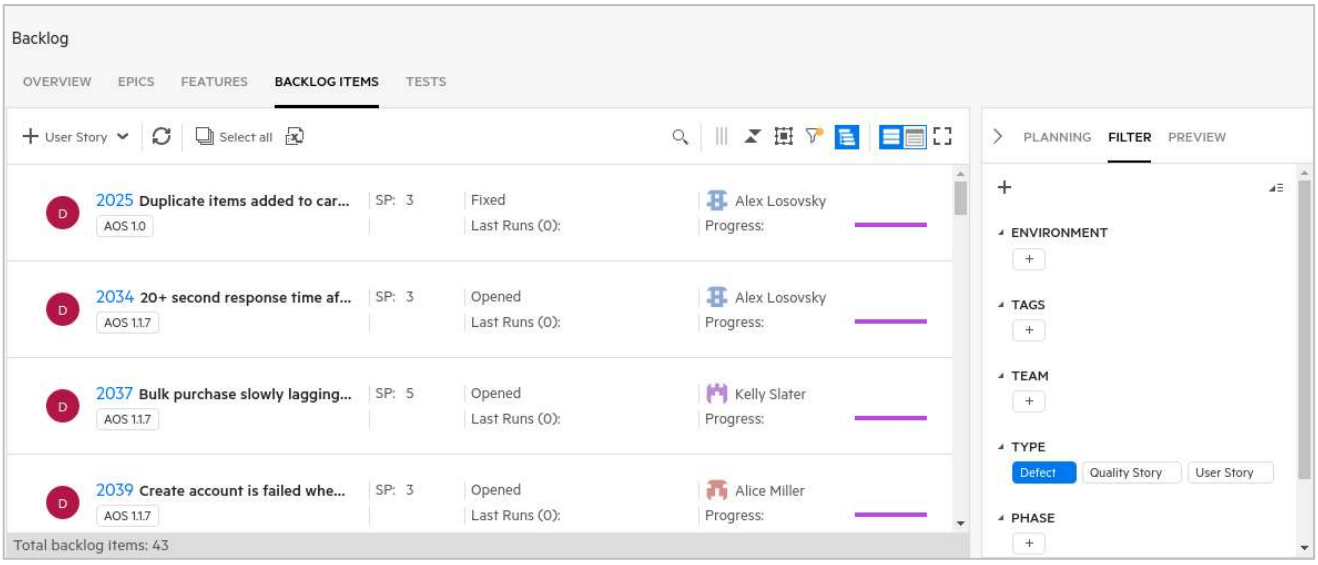


11. Select the **Filter** tab on right side bar.

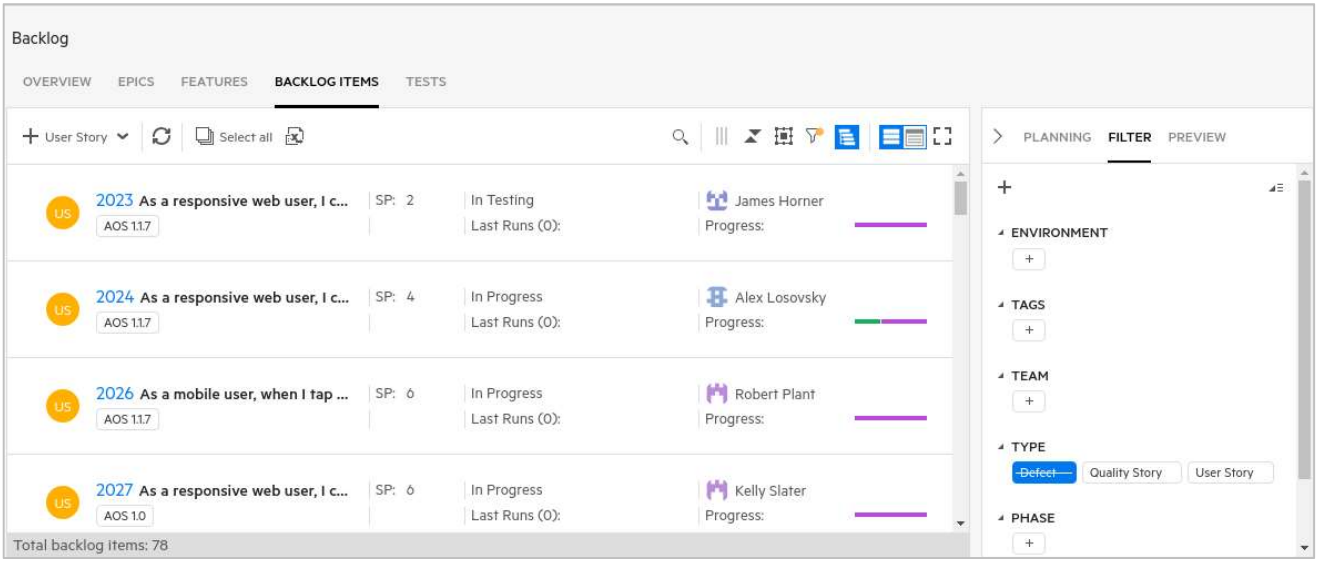


12. In the Filter section, click **Defects** to filter and only show items of type Defect.





13. If you want to filter *negatively*, **right-click** on the item type to de-select it. The example below will show everything except defects.

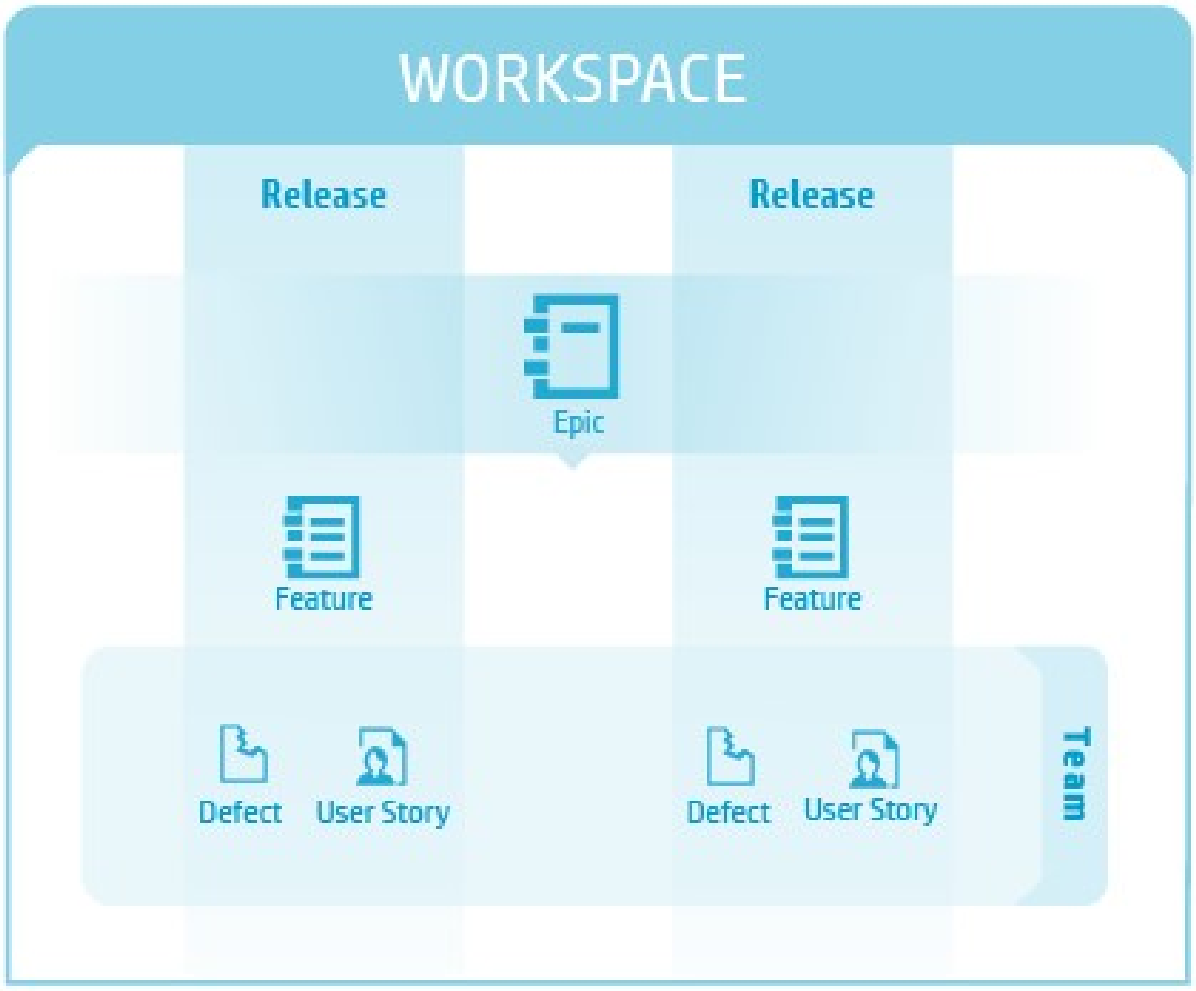


14. Clear any filters before proceeding (they should not be blue).

Exercise 5: Building an Agile Backlog

In this exercise you will work on understanding the ALM Octane agile methodology by populating the backlog with some epics, features and user stories.

Backlog Structure/Hierarchy:



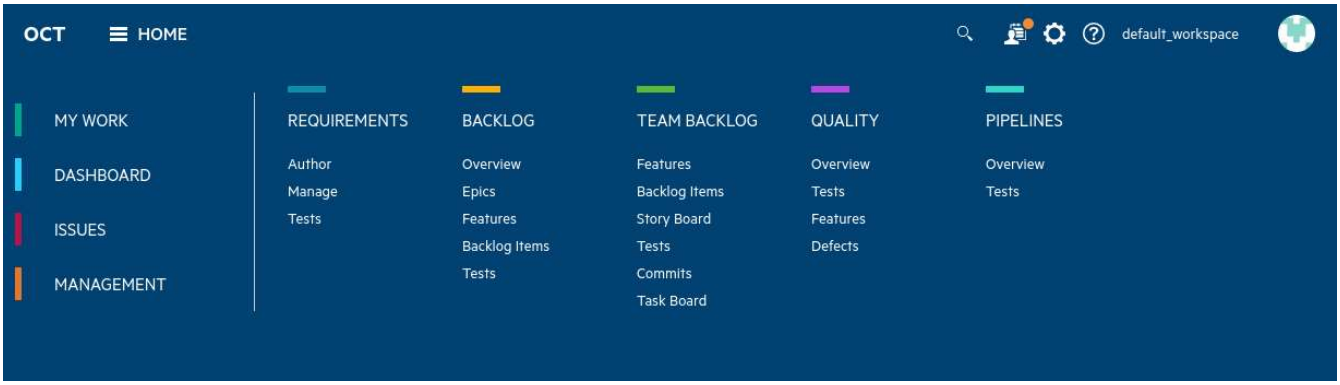
User stories (US) form the basis of what the system does. These are abstracted up into Features which are services that fulfills a stakeholder need. Features are further abstracted into Epics which is even a larger chunk of work that has one common objective.

The backlog consists of the following agile entities:

Releases	A release is a group of application changes made available to your users at the same time that will be distributed as a completed version of the application at the end of the release. The release provides the framework around which you organize the backlog items. Releases can be divided into sprints or milestones.
Epics	The top-level objectives your product must meet, or the product's high-level functional areas. They are large components or aims for your product or project. Epics can be and are implemented across releases. Epics are divided into features.
Features	A deliverable unit or mid-size functional goal of the application which is utilized by users. These areas are typically implemented within the scope of a release. They are groups of actions or functions that you plan for your product or project. A feature is divided into user stories, defects, quality stories and tests.
Backlog Items	The functional parts of a feature. Stories include user stories, quality stories, and defects and are assigned first to a team and then to an individual developer. Typically they are implemented within a sprint cycle.

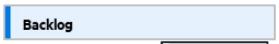
Navigating through the Modules

1. Switch back to the Octane tab in your browser
2. Click the **Modules Icon** to see the Octane modules.
    a. The Octane triple bar icon allows you to navigate through the different modules, including My Work, Dashboard, Issues, Management, Requirements, Backlog, Team Backlog, Quality, and Pipelines.



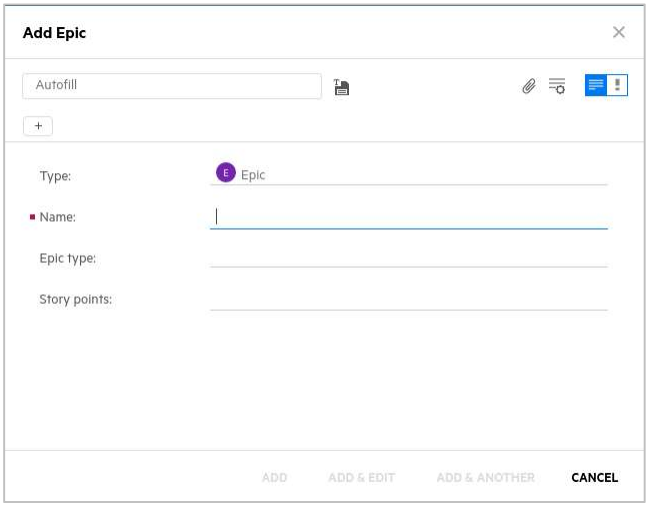
Creating an Epic

Select the **Backlog Module** 

1. Select **Backlog** tree title on the left side of the screen. 

Select the **Epics tab**, and then select the button to add a **new Epic**. 

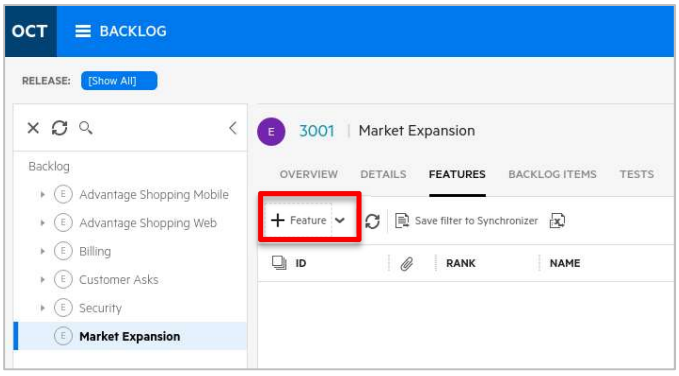
    a. Epic Name: **Market Expansion**
    b. Epic Type: **Business**
    c. Assign Story Points: **50**
    d. Select **Add**



Creating a Feature

1. Select the Epic you just created from the left side tree list.
Select the **Features Tab** and click the 

button to add a **new Feature**.



a. Feature Name: **Home Security**

b. Epic: **Market Expansion**

c. Feature Type: **Business**

d. Assign Story Points: **20**

e. Release: **1.1.7**

f. Select **Add**

NOTE: Story Points Discussion: At this point we have created a new Epic (Market Expansion) and Feature (Home Security) that we wish to implement. Next we'll drill deeper and create a user story and the tasks required to complete that user story. Story Points are a rough estimate of the effort it will take to implement it. For our example, consider one story point equals half a day (4 hours) of effort.

Creating a User Story and Tasks

1. Select the Home Security feature that you just created in the left side tree. Select the **Backlog Items** tab and select the + User Story button to add a new **User Story**.

   a. User Story Name: **As a user I want to purchase a security camera**

   b. Application module: **15 Security**

   c. Release: **< CLEAR VALUE – empty >**

   d. Team: **AOS_Infra**

   e. Assign Story Points: **10 – DO NOT CLICK "ADD" YET!**

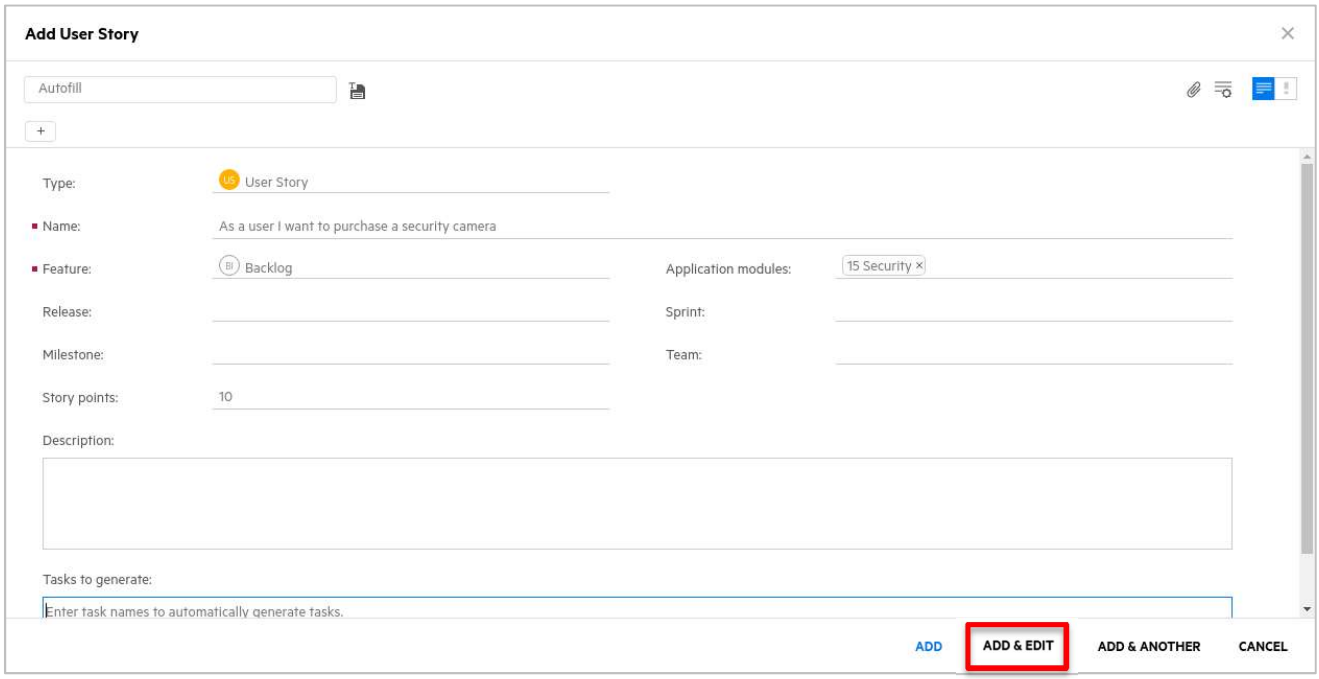
> **What's an Application Module?**
> An application module is a segment of the product's functional areas. In our example of an online e-commerce site, Security contains items that keep our customer safe. While our top-level categories don't cover Security devices (the Security application module is concerned with the website's security) we can begin to market a home security system as a Security item and then expand this into a new Security Devices application module later.

2. **Scroll Down** to see the **Tasks to generate**: section.

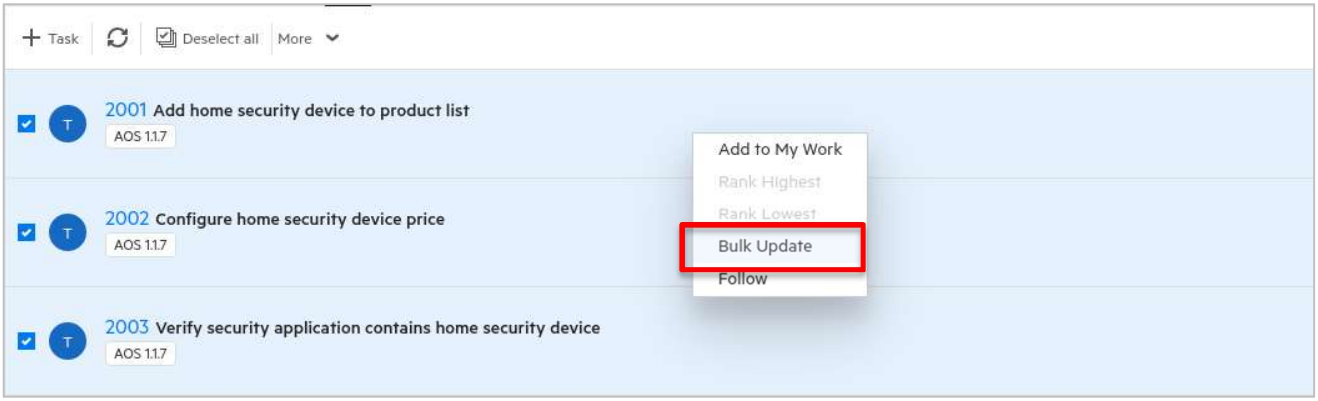
3. Add **Tasks** by typing the task names in the **Tasks to generate** box and hitting **Enter** after each task (see image below).

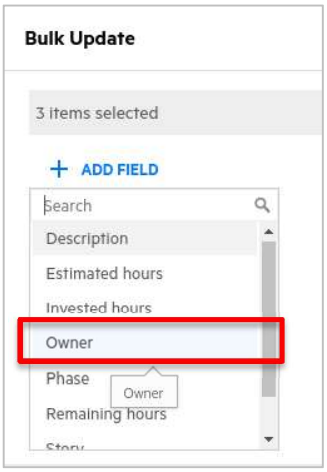
   a. **Add home security device to product list**
   Configure home security device price
   Verify security application contains home security device

   b. Select **Add & Edit**



4. You'll now see the details on the user story (it automatically created our three tasks).
5. Set the Owner (Select user circle on the right) to be **sa@nga**
6. Select the **Tasks** tab so we can set the Owner for these three tasks.
7. Select the Smart List View on the right and select all three tasks.
8. Select **More | v -> Bulk Update** and add a field (Owner) and set the owner to be **sa@nga**. You can also just right-click on the selections and choose **Bulk Update**.

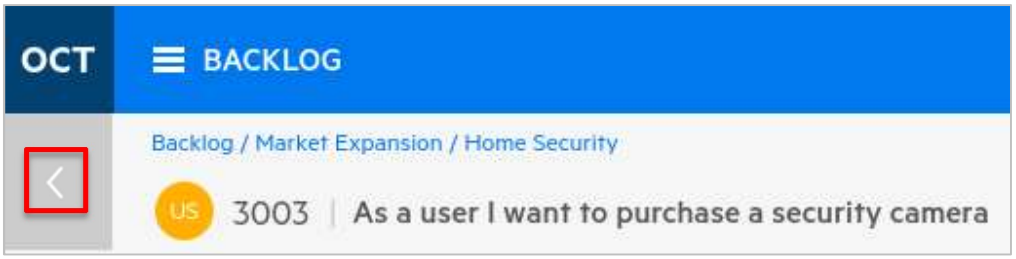


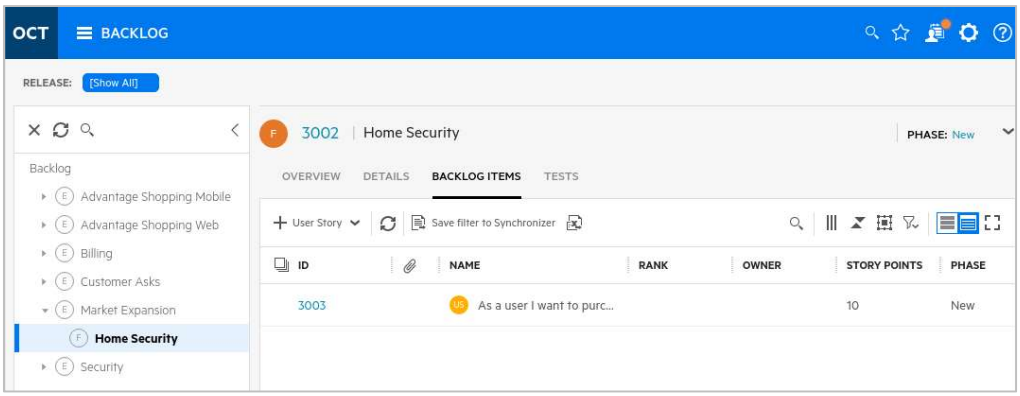


Bulk Update is very useful in situations where you need to change one field on a variety of objects. Often you may want to select a number of tasks or user stories and assign them to a specific team or individual.

9. Verify that all three tasks are now owned by **sa@nga**.

10. Click the **back** arrow to get back to the **Home Security** user story.





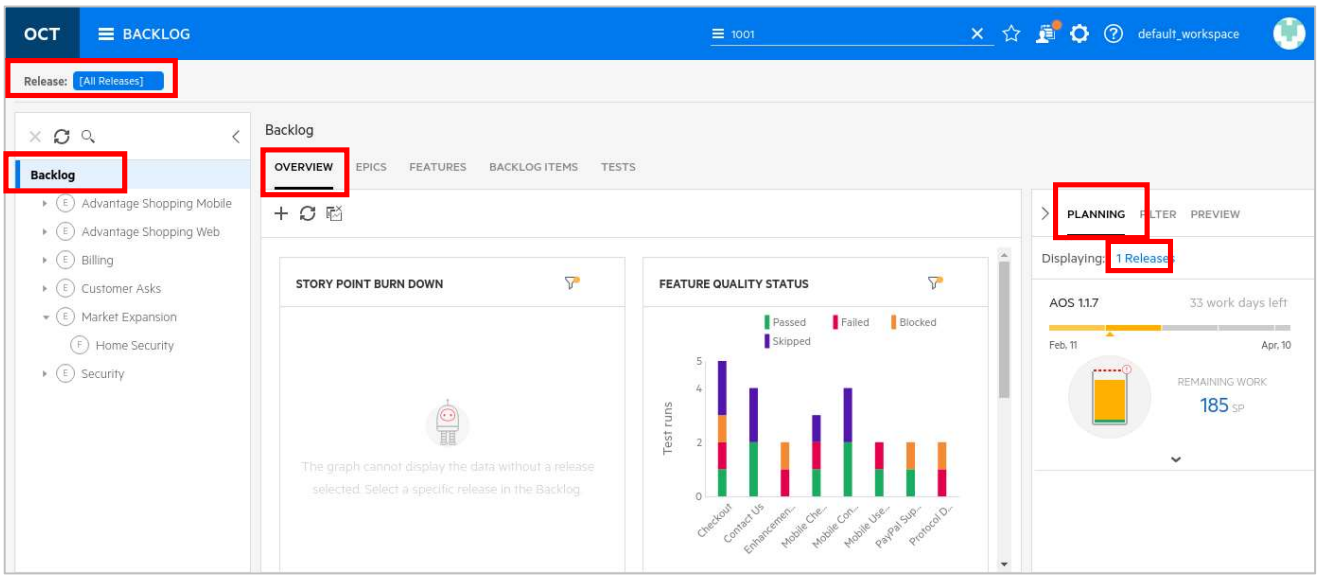
What have we just done?
We started building some new functionality for our application in development. We began by defining a high-level epic called **Market Expansion** and then defined a feature called **Home Security**. Then we built a user story (these are real guts of an agile project) that requested the ability to buy security cameras. This request will necessarily involve multiple tasks to be completed so we added those as well.

Planning a Release

Releases and Sprints can often be confused. A Sprint is a potential release but often Sprints don't end up in a release. Instead, think of a Release as a stable build that can be released.

1. Select the entire **Backlog** on the left side tree. 

2. Set a release filter for the **Overview** tab (set to **[All Releases]**).
3. Set a release filter for the right side **Planning** tab (set to **AOS 1.1.7**).
4. Select the **Overview tab** to view the custom Graphs and Widgets for the backlog.



*** Here you can see a dashboard for the backlog that will display data to help teams plan and assess quality, and progress throughout a release.*

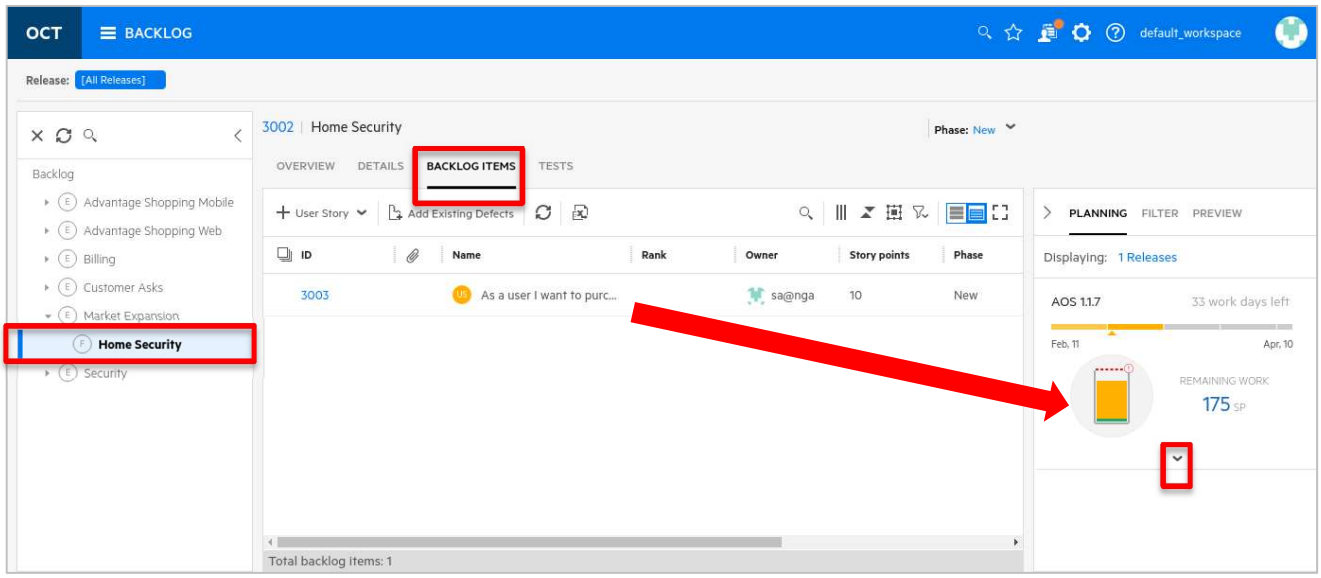
To see the features of Octane's Release Planning:

5. Assign your User Story to the 1.1.7 Release by selecting the **Feature** you created (**Home Security**) from the left navigation tree and then select the **Backlog Items** tab. On the right hand side select the **Planning Tab**. The Planning tab contains the release's planning bucket:

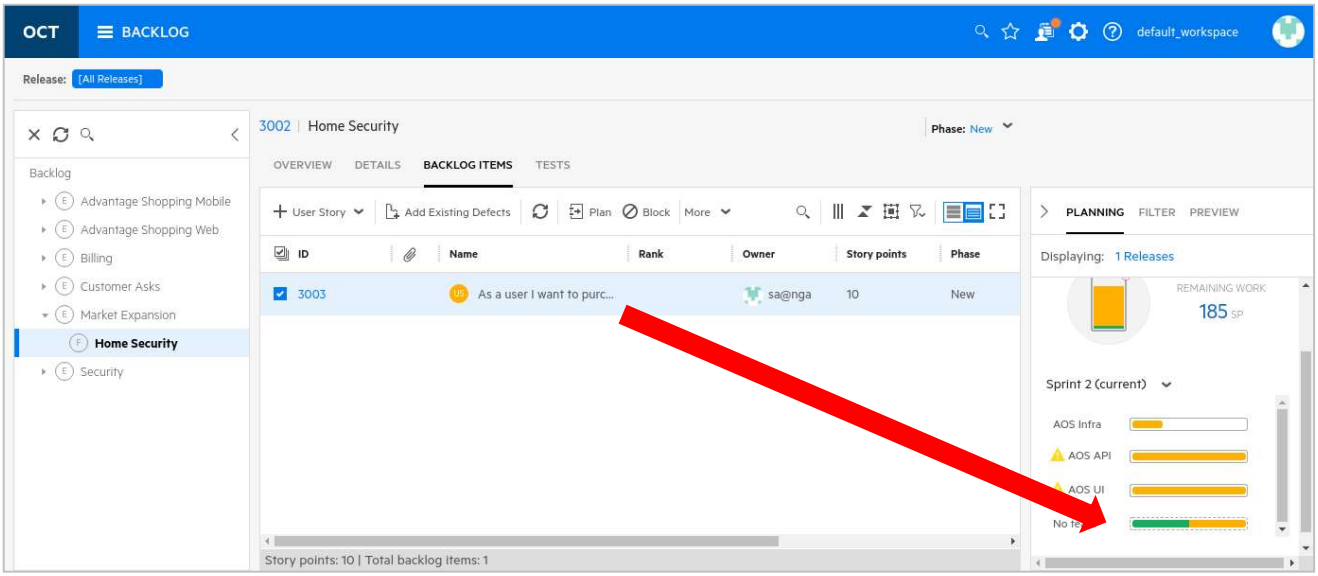
    a. Click the link next to **Displaying:** and verify it shows **1 Releases**.

    b. Select the **User Story** you created and **drag and drop** that item onto the **AOS 1.1.7 Release**.

***You will see the release bucket update to include the user story and story points you just assigned to it.*



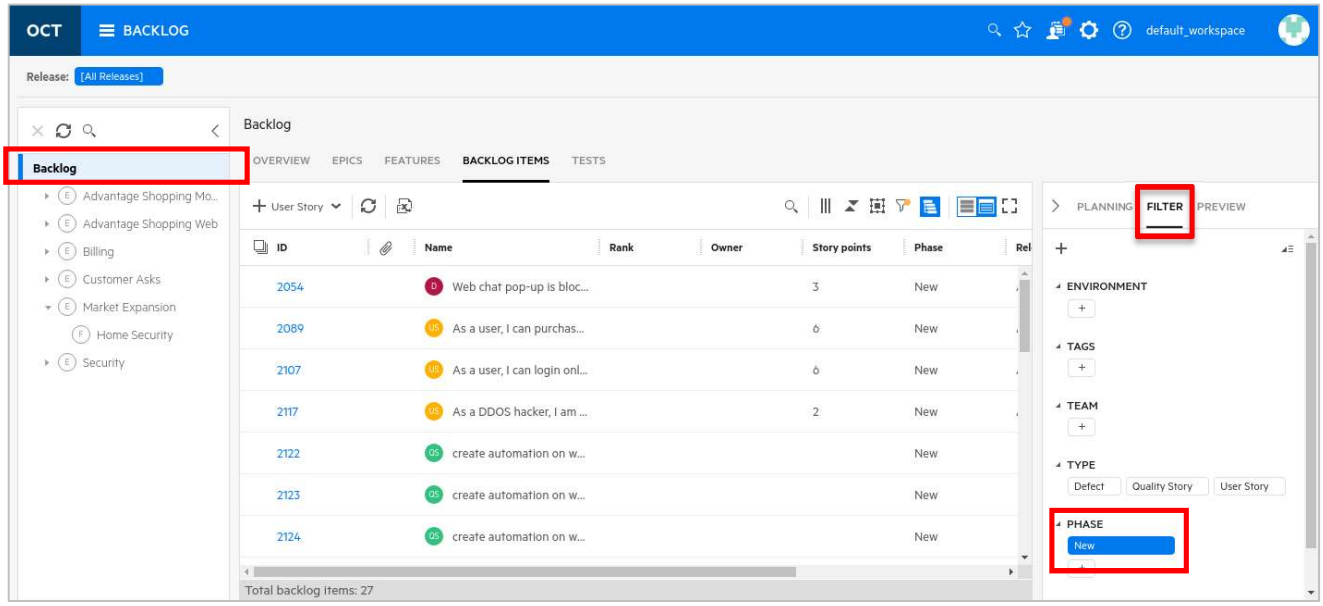
6. Let's also assign the User Story to a team. Expand the **Planning Bucket** to view the teams associated with the Release. Click and drag the backlog item to the **AOS Infra Team**.



To filter by various properties for a Backlog Item:

7. Again, select the entire Backlog on the left navigation. 

8. Select the **Filter** tab on the right hand side. Filter the backlog using the out of the box generated tags. Select the **New** tag under the **Phases** category. This will filter the backlog items and only display those that whose phase is New.



***Tags are user-defined identifiers and are used to label an item in a meaningful way for a user or an organization. Tags are used as a label or flag on an item and help when searching or sorting similar items or topics.*

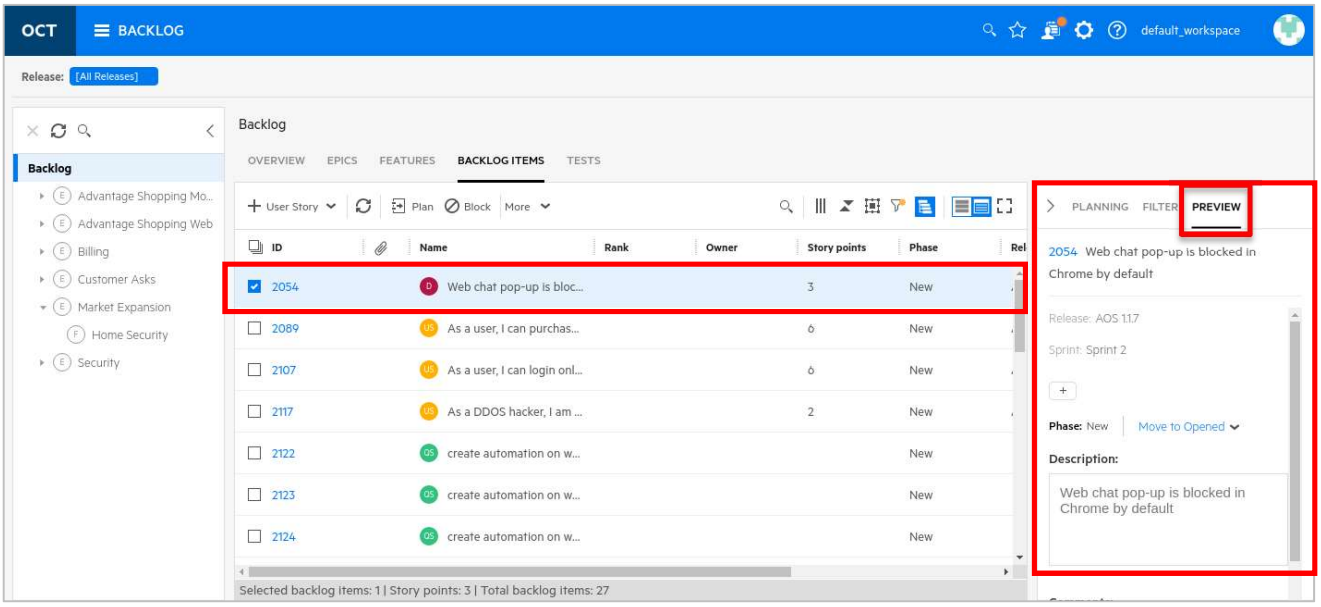
NOTE: Since you can filter in various ways (Filter icon, Filter panel) and these settings are not always obvious, you can look at the filter icon to see if your data is currently being filtered.

If it has a yellow circle by it, your data is being filtered 

To **Preview** items from the Backlog module:

9. Select the top row in your **Backlog Items** list.
10. Select the **Preview tab** on the right hand side.
11. From here you can quickly edit the details of your backlog item.

*** If you want to gain a high level understanding of a backlog item you can select the entity and view the Preview tab. This will show you a description of the backlog item, the Release and Sprint it is assigned to, and Comments to other team members.*



Awesome!

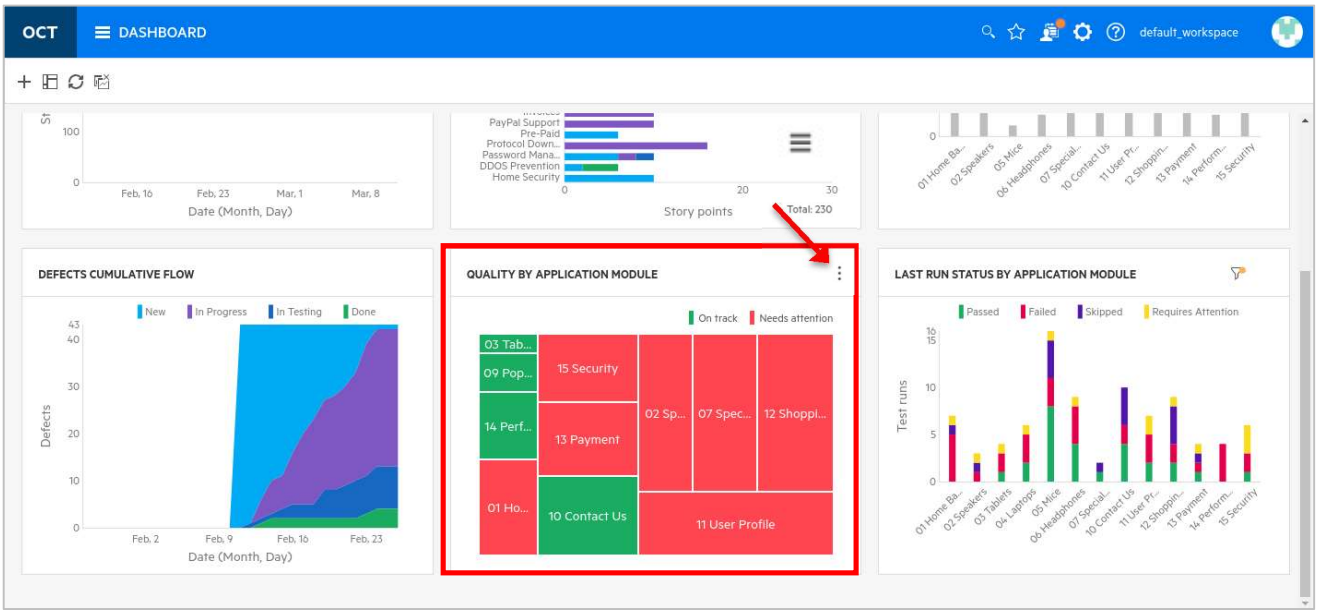
You have just built a nice agile Backlog.

Exercise 6: Defining Test Coverage and My Work

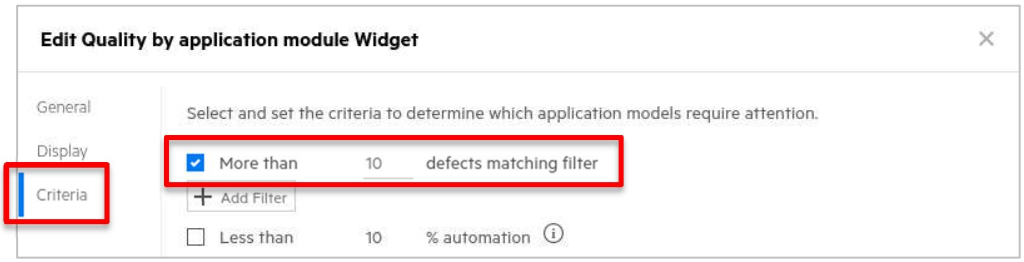
One of the chief advantages in ALM Octane is its ability to identify where issues have occurred and to locate and prioritize work to bring the project back on track. Octane also houses a My Work module that filters work assigned to an individual that they can focus on tasks they need to complete.

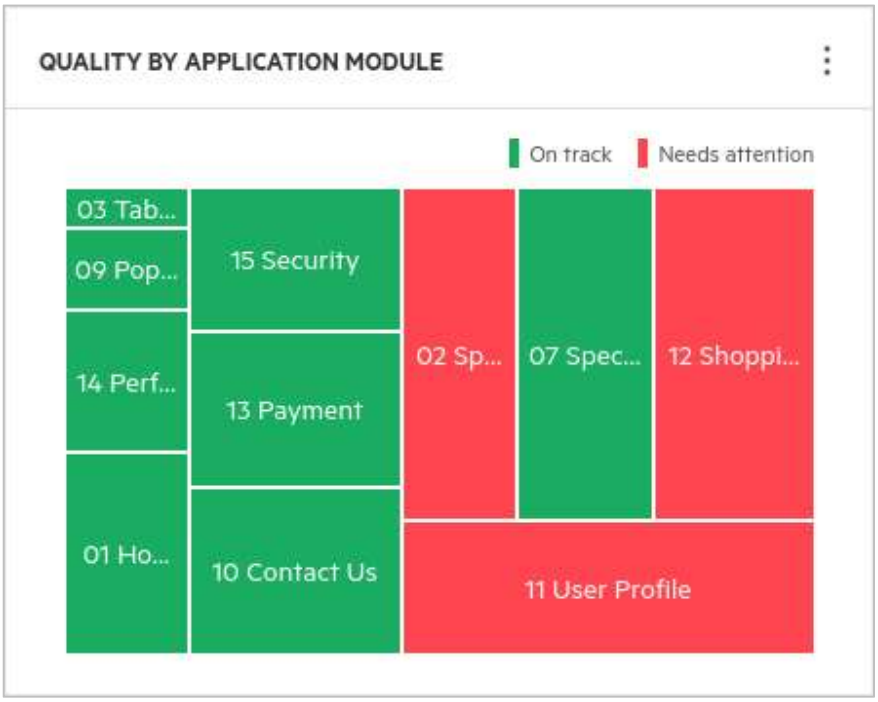
Viewing Test Coverage in the Dashboard

1. Navigate to the **Dashboard Module**. 



2. Scroll down and select the **Quality by Application Module** graph. This graph shows a heat map of the application modules with respect to which modules are more problematic based on our criteria. The current criteria says to mark any module as red if the associated defect count is greater than 5 defects.
3. Mouse over the three dots and select **Configure** to change the criteria.
4. **Change** the **Criteria** to be **More than 10 defects matching filter** and **Save** your change.



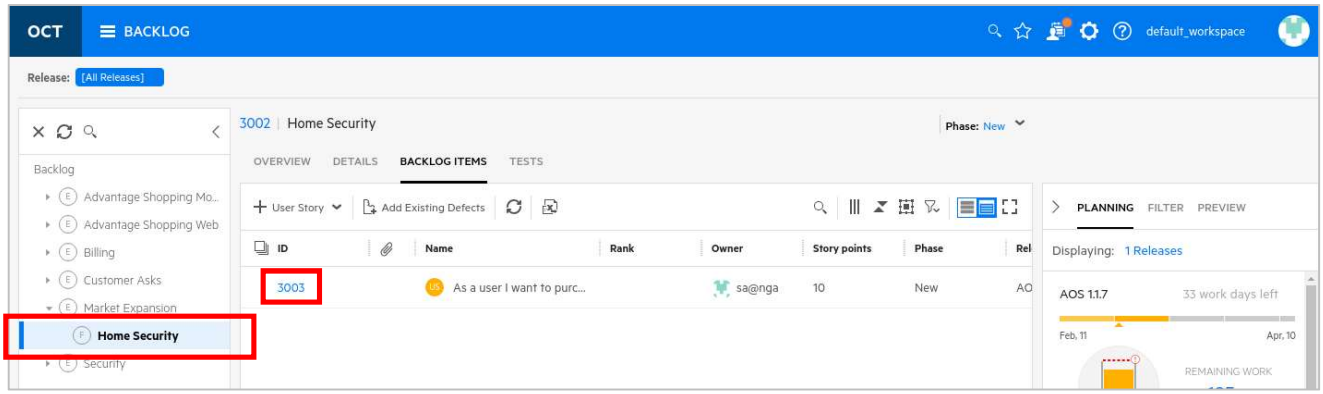


*** The Quality by Application Module graph displays the overall quality of your application. We've changed the criteria to be more forgiving (up to 10 defects is acceptable) and thus we see more green areas.*
To increase the quality of our application we will create a test to cover the Application Online Store module.

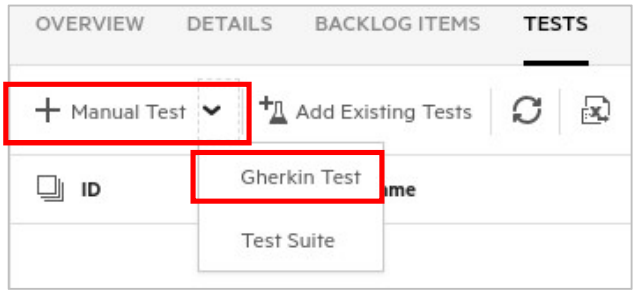
Creating Test Coverage in the Backlog

1. Navigate to the Backlog Module. 

2. Select and click into the **User Story** numeric link you created in Exercise 1 (under **Home Security**).

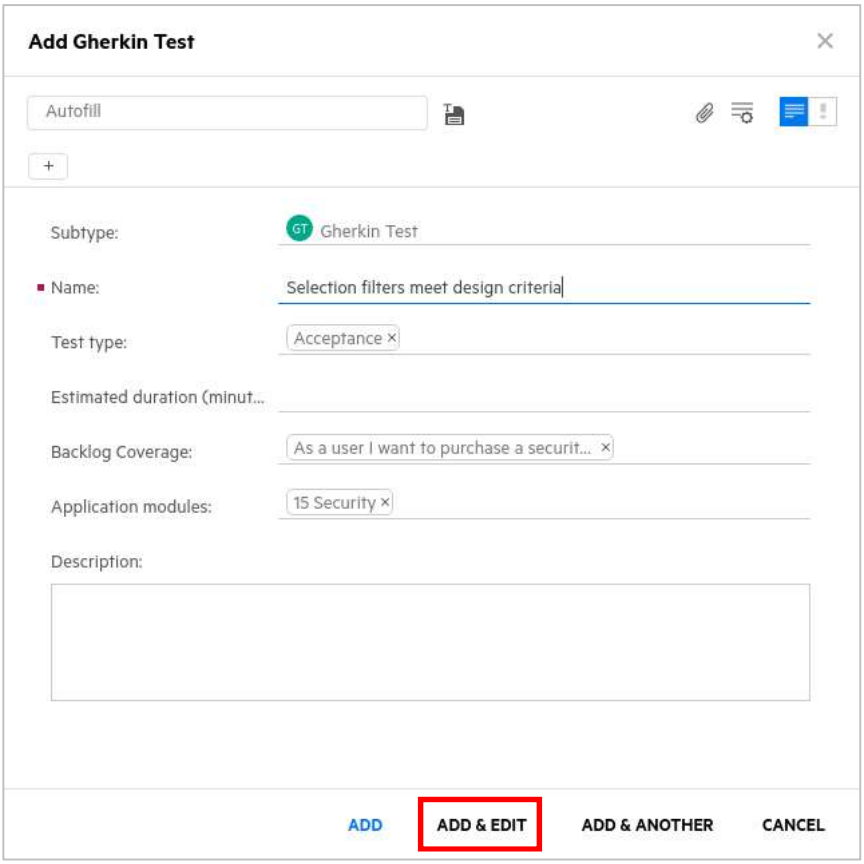


3. Select the **Tests** tab and select the pull-down next to **+ Manual Test** to choose to add a new **Gherkin Test**.



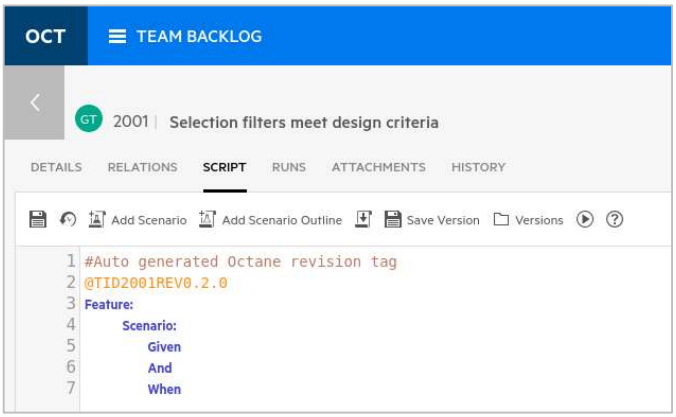
*** Gherkin is a well-known syntax for writing tests using a behavior-driven development (BDD) methodology. This methodology enables you to test using both the developer and user perspectives as you create scenarios of use instead of individual test steps. Gherkin syntax uses plain-text English with a specific structure. It is designed to be easy to learn by non- programmers, yet structured enough to allow specific examples of how the application is used in the real world. When you perform the test, you follow the scenario and use the parameters to perform the test.*

4. Fill in the details of the Gherkin Test.
    a. Name: **Selection filters meet design criteria**
    b. Select **Add & Edit**



5. Select the **Add Scenario** button 

This will add a Gherkin template to define a test scenario.

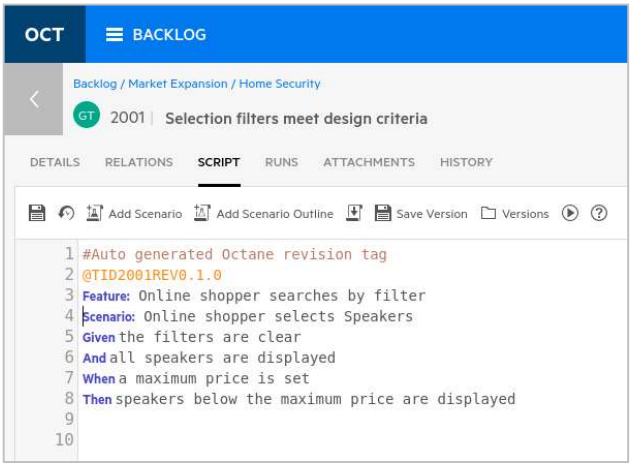


6. Copy the following text and paste it into the scenario section (overwriting the existing Blue text). This creates a test using the Gherkin syntax (the reserved words will automatically be in blue).

   Feature: Online shopper searches by filter
   Scenario: Online shopper selects Speakers
   Given the filters are clear
   And all speakers are displayed
   When a maximum price is set
   Then speakers below the maximum price are displayed

7. Click the **Details** tab and set the **Owner** to be **sa@nga**.

8. Click **Save**. 
   This saves our Gherkin-syntax manual test. This test can actually be automated later and the automation uses these English-language terms as function names.

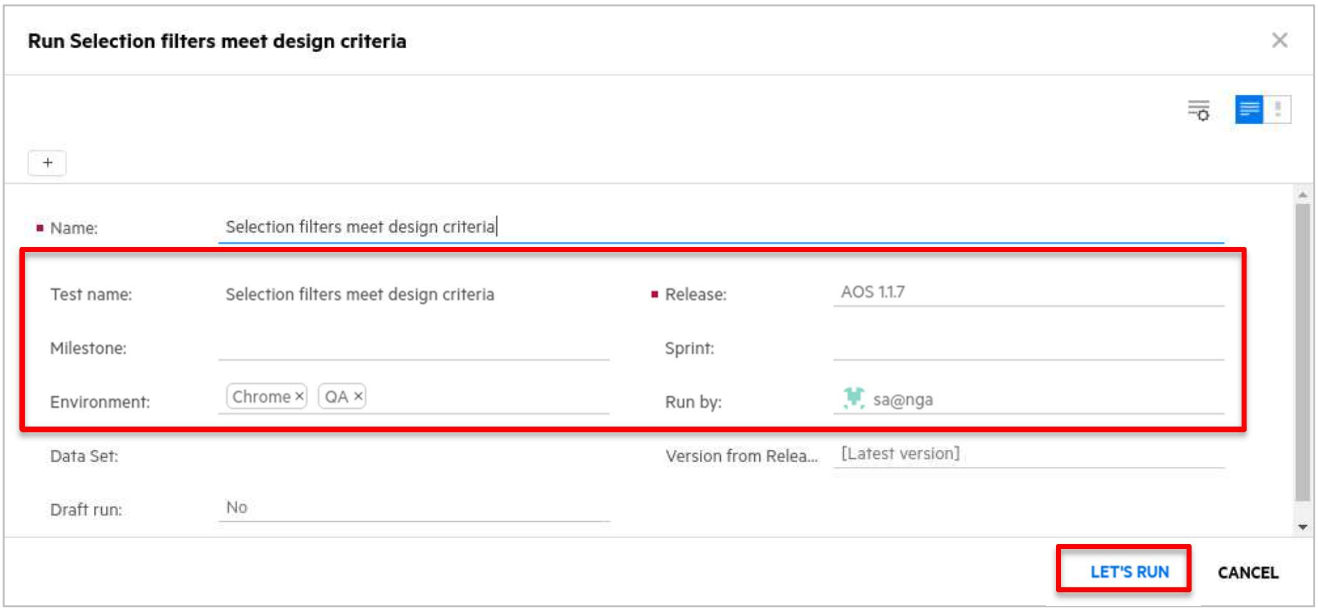


Executing a Standard Manual Test

1. Select the **Run test** button  to launch your manual Gherkin test.

   Release: **AOS 1.1.7**
   Environment: **Chrome + QA**

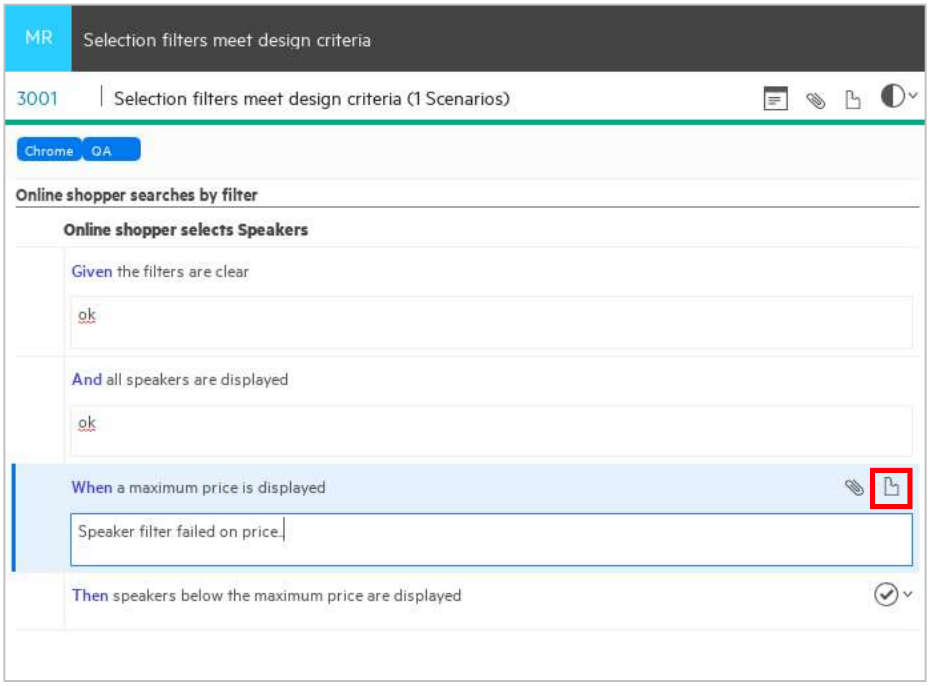
2. Click **OK.**
   Select **"Let's Run!"**



*** In ALM Octane your manual tests are run using the ALM Octane Manual Runner. When you open the test in Manual Runner, perform each step as written in the test or scenario. Then enter actual values for each step or scenario and assign the appropriate status to the validation step, such as Passed, Failed, and so on. If necessary, report defects and add attachments to help others understand the test results.*

3. We're going to simulate running this manual test and then pretend to discover a defect in our application.

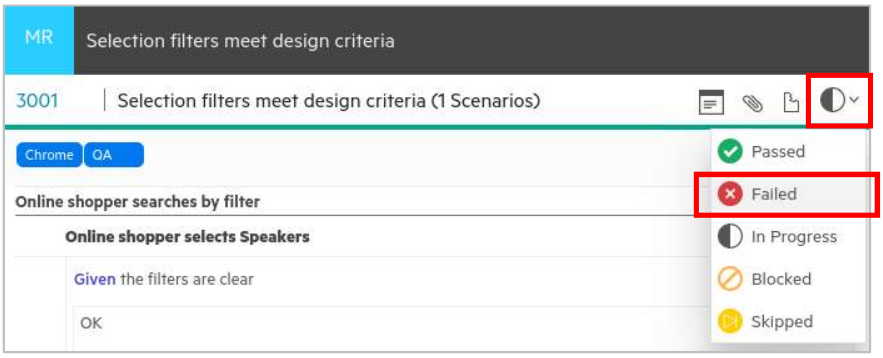
   In the Manual Runner interface, enter OK for the first two steps and then enter Speaker filter failed on price into the third step as shown below and then select the Create Defect icon.



4. Create a **defect** during the test run on the third step (during the **When** step).

   a. Name: **Speaker filter failed on price**
   b. Feature: **Home Security**
   c. Severity: **Critical**
   d. Application: **15 Security**
   e. Release: **AOS 1.1.7**
   f. **Defect Type: Pre-release**
   g. Click **Add**

5. Once you have added the defect set the status of the manual run to **Failed**.

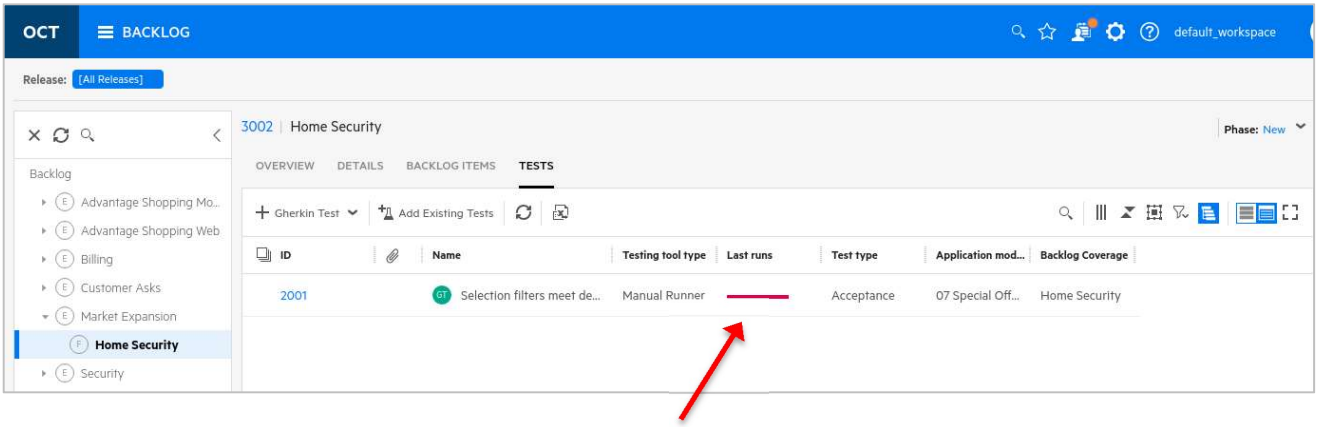


6. Click  at the bottom of the manual runner to stop the run and record the failure.

What did we just do?

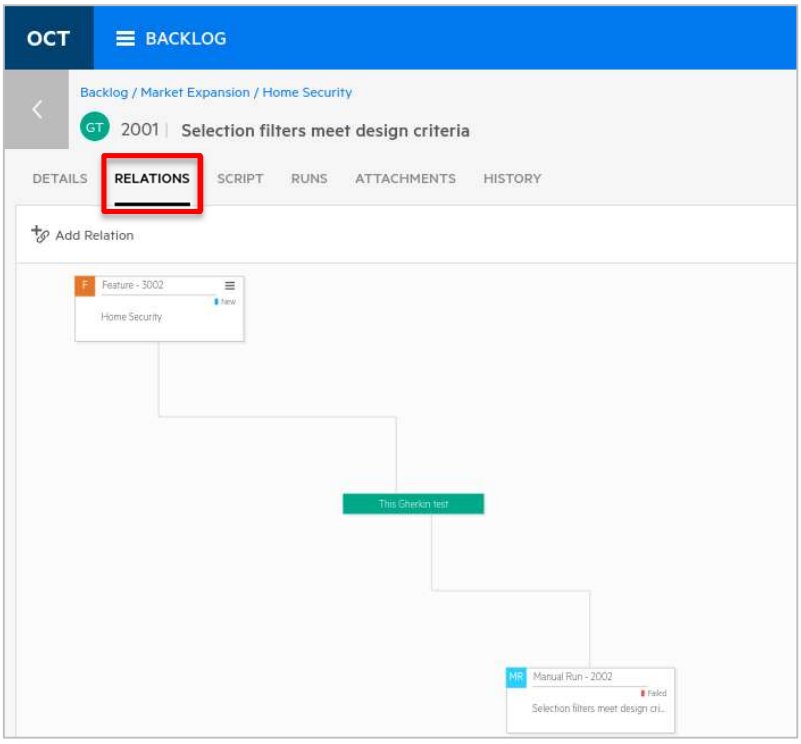
We created a manual test – the most prevalent kind of testing – and then ran that test. We used the Gherkin syntax, a loosely structured test language as a textual template. During the run, we simulated a defect in one of the test steps and immediately created an Octane defect and then returned to our manual test run and failed that run. This represents a typical scenario that might happen when defining and running manual tests.

7. Select the **Tests** tab and verify, by the red color under **Last Runs**, the failed test result.



Reviewing Relations, Test Runs, and Automation Status

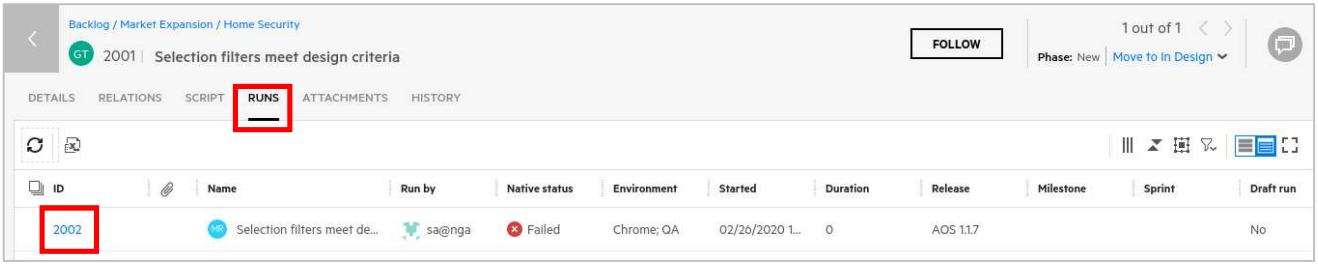
1. Select the **RELATIONS** tab. Use your mouse wheel to zoom in and out and to reposition the entities so they are easier to see.



*** A relation represents direct links between entities. Relations show traceability between epics, features, user stories, tests, test runs and defects. To create traceability between your test and another User Story, Requirement or Feature you can select the Add Relation button.*

Relations are also immediately created during certain activities like creating a user story under a feature or creating a defect while running a test. ALM Octane is very good and understanding these relationships and creates an audit trail of everything that happens to these entities.

2. Select the **Runs** tab to see all the test runs and their data.



*** You can also adjust your view to see a **Grid View** or **Smart List** using these controls:* 

3. Notice the round circle icon with **GT** in it. This represents a **G**herkin **T**est.

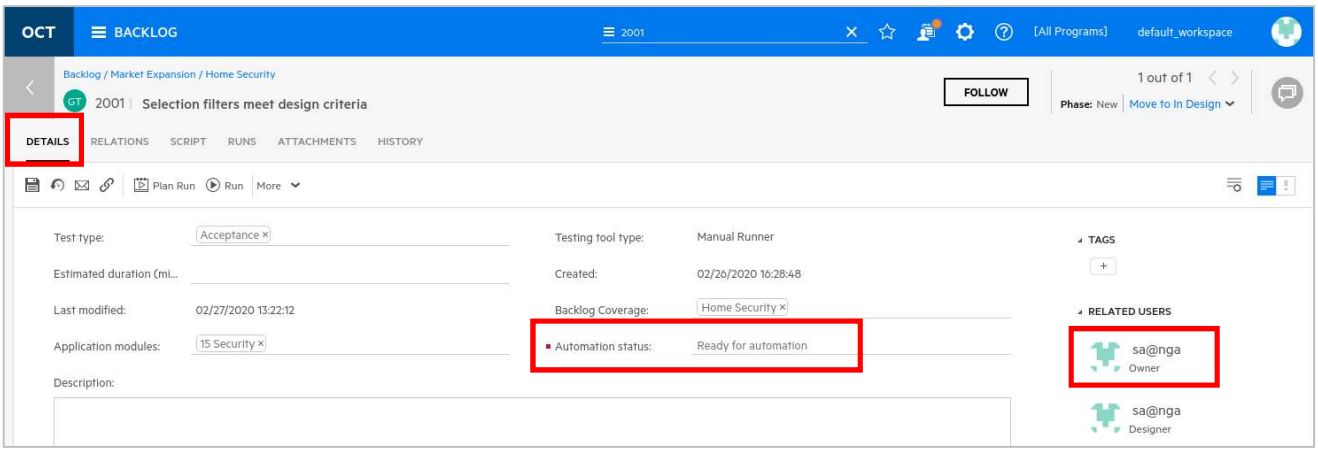
   Almost every item in Octane follow this pattern so you know what kind of object you're looking at. Can you think of an entity that doesn't have an icon? _____

4. Select the **Details** tab and set:

   a. Automation Status: **Ready for Automation**

   b. Owner: **sa@nga**

   c. Click **Save**



Managing the Team Backlog

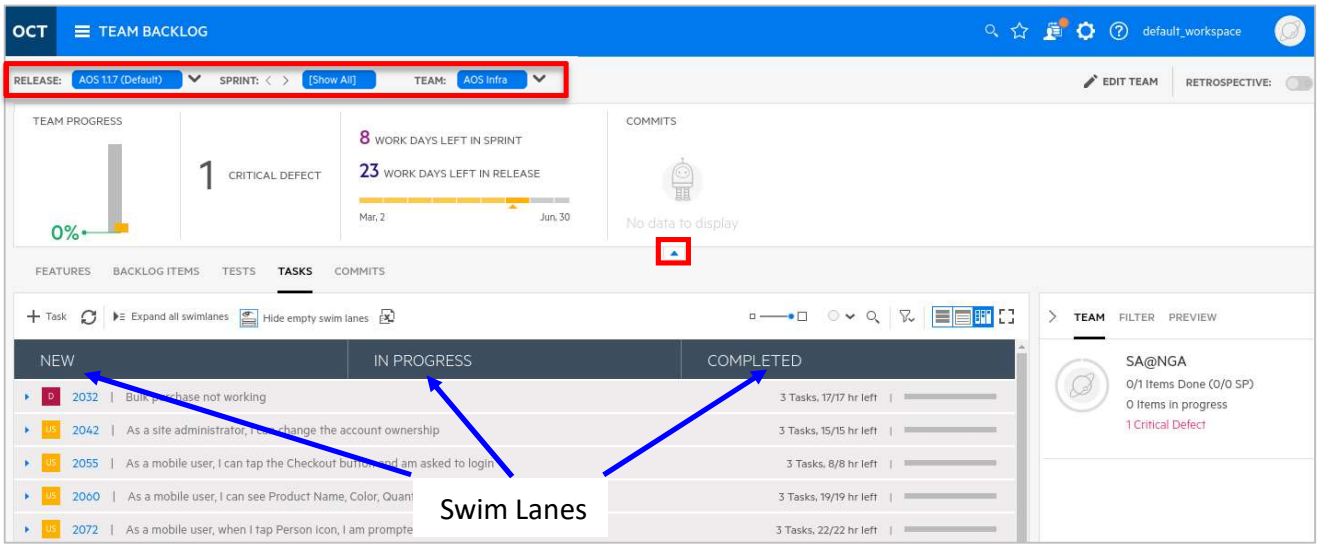
After setting up the overall release plan and work in the Backlog, it is important to demonstrate progress by working with your team's backlog items to start carrying out your development plan.

1. Select **Team Backlog** 

from the modules menu.

2. Set the:

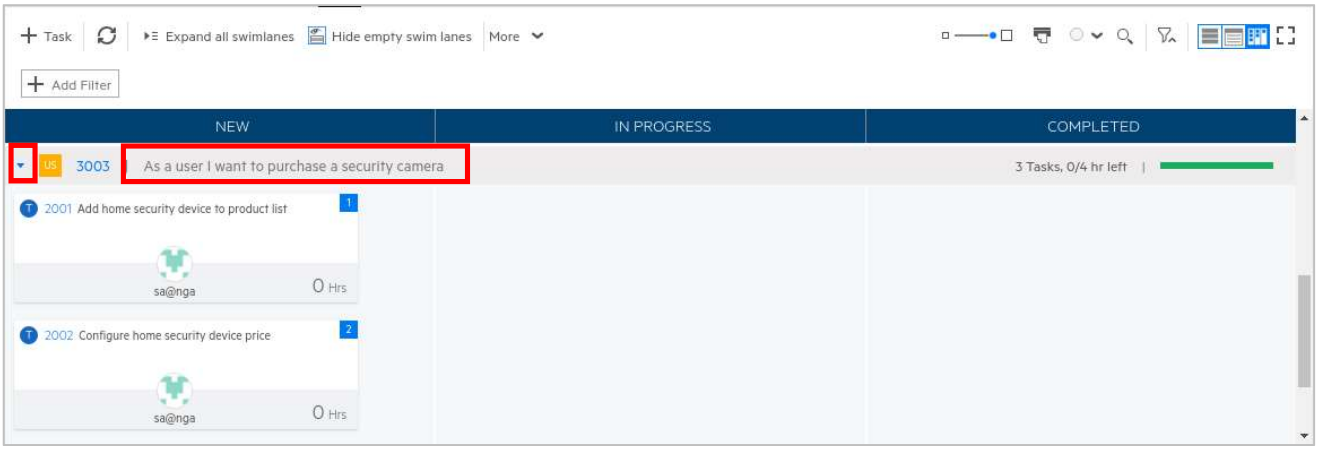
   - Release to: **AOS 1.1.7**
   - Sprint to: **Show All**
   - Team to: **AOS Infra**
   - Tab to be: **Tasks**



Notice that Octane even provides a way to shrink this sort of view so it takes up less space (see red box above).

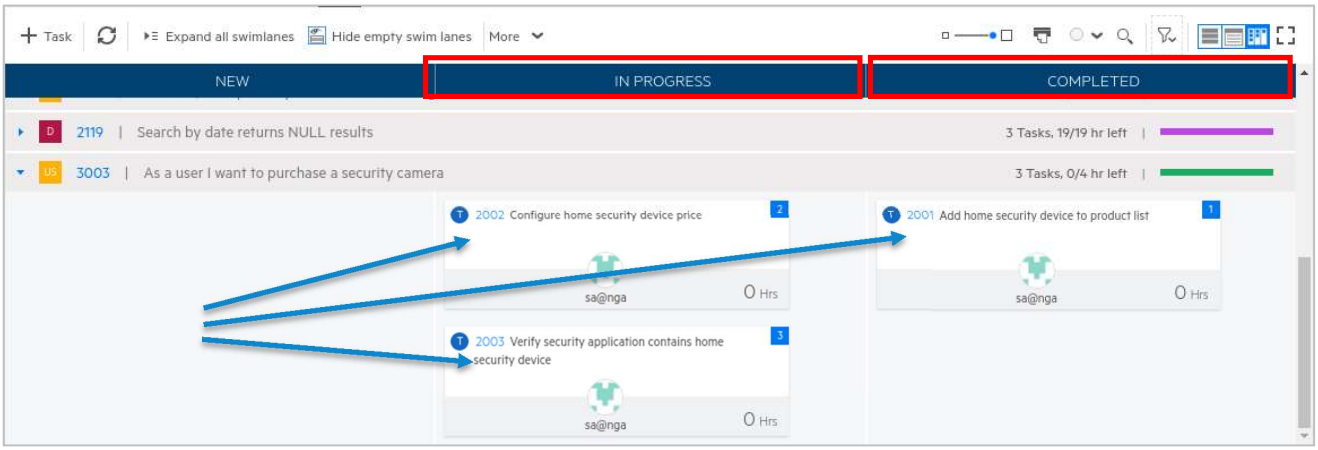
During the development period, it is important to track individual progress to make sure all parts of the team are progressing toward the stated sprint or release goals. You do this by moving tasks through the swim lanes.

3. Select the **Tasks** tab and scroll all the way down to the user story you created previously. Expand the user story titled “As a user I want to purchase a security camera” to see the tasks.

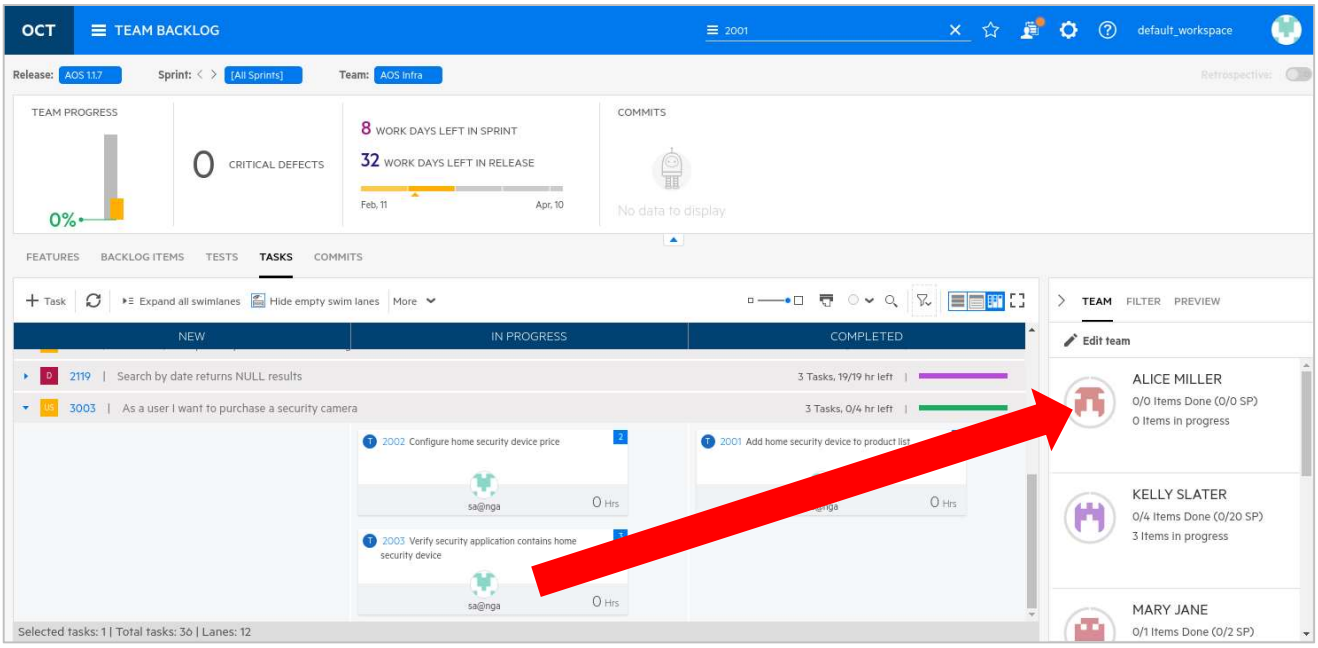


***In the upper left corner of the Team Backlog module (the sprint summary area), you can view a graph that displays the overall development status in the selected sprint(s). This graph displays the expected and planned velocity in story points. In addition, it also shows the number of critical defects assigned to the team. The progress is measured by the story points assigned to all members of the team in the current sprint cycle.*

4. Move tasks (using drag and drop) from your User Story created through the swim lanes to track progress. Move them into the positions shown below.



5. Next, we'll assign a task to a new team member. Select the **Team** tab in the right hand pane and click and drag a task to the team member. Drag the Verify task to Alice Miller.



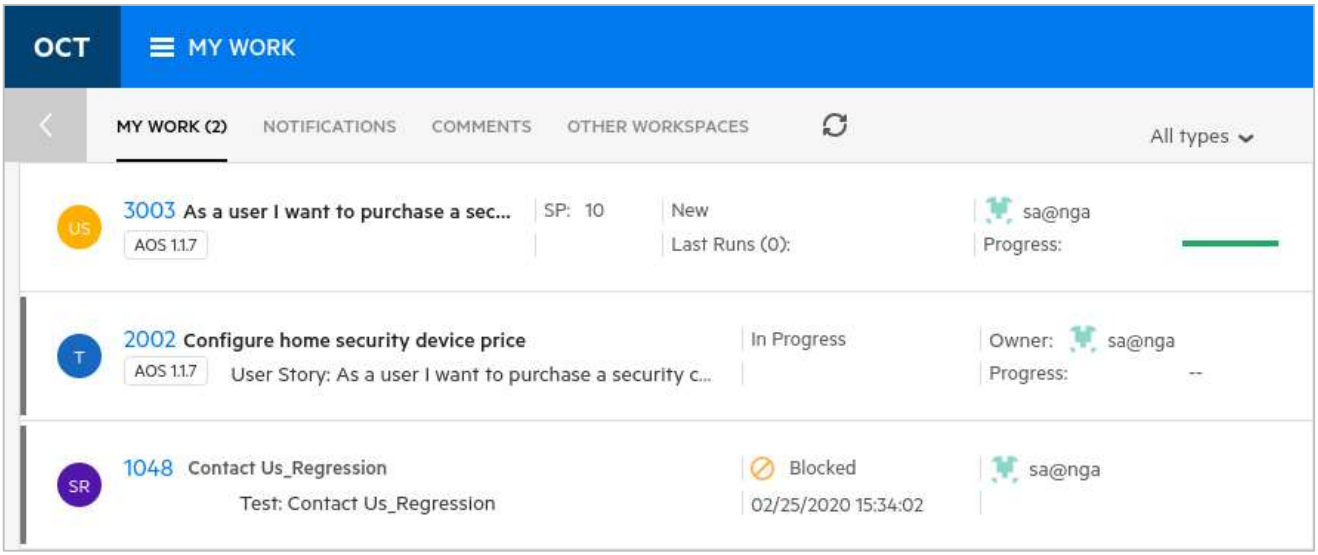
6. You should see the task change owner from sa@nga to Alice Miller.

Reviewing My Work

Now that you have planned all the work to be completed in a release, you may want to track your work with a personalized to-do list of work items. The items that appear in the **My Work module** are defined by the ALM Octane administrator. You can customize this as needed in the Workspace settings. Select the **My Work module** located in the menu bar. This module enables you to quickly view and complete the work assigned to you.

1. Select the **My Work** module through the icon  on the Octane module menu bar.

*** By default, items are displayed in the My Work page based on the owner of the item. You should be able to find the User Story, Defect, Gherkin Test and Task that you created previously. You can also filter these items by type.*



After viewing your work in this module you would typically click the numeric link to drill into that item and begin working on it. Your work items can be moved out of the My Work module by a number of means:

1. Select the item and in the right hand pane, click **Dismiss**. This marks the work as having finished. This also works for any item type in the My Work module.
2. Drill into the item and set the owner to be someone else. This removes it from your work and assigns it to someone else.
3. For Tasks, move the item into the Completed phase. This also marks it as finished.

Challenge Question: You created three tasks but only one is shown here – why?

One task was reassigned to Alice Miller and the other was moved to the Completed phase.

This table below shows the types of items listed in **My Work**.

Assigned items	Assigned items may include user stories, defects, quality stories, or tasks. By default, these items will be listed if they are in a phase that belongs to the **New**, **In Progress**, or **In Testing** metaphases. If your tasks are included in a user story not assigned to you, the task item contains the name and a link of the user story.
Tests	Tests are listed if they are in a phase that belongs to the **New**, **In Design**, or **Awaiting Revision** metaphase or the **Requires update** phase of a test awaiting automation. You can change a test's phase or add comments. Test descriptions are view-only.
Test runs	Test runs assigned to you will be listed if they are in **Planned**, **In Progress**, or **Blocked** status. Passed, Failed, or Obsolete runs will not display. Test run releases, environments, and run descriptions are view-only and cannot be changed.
Mention in comments	Comments addressed to mentioned users are displayed in the **Comments** tab. The mentioned users can click the relevant comment to view the full list of comments related to a particular item and can respond.
Following	When you select an item to follow (from within the item itself), it is displayed in the list of **Notifications**. You can view relevant details and progress of these items easily from this list.
Other users items	If you right-click an item in the Backlog module and select **Add to My Work**, ALM Octane displays this item in your My Work list and enables you to edit it.

Great Job!

You have just learned a ton about test coverage and My Work.

Exercise 7: Managing Quality

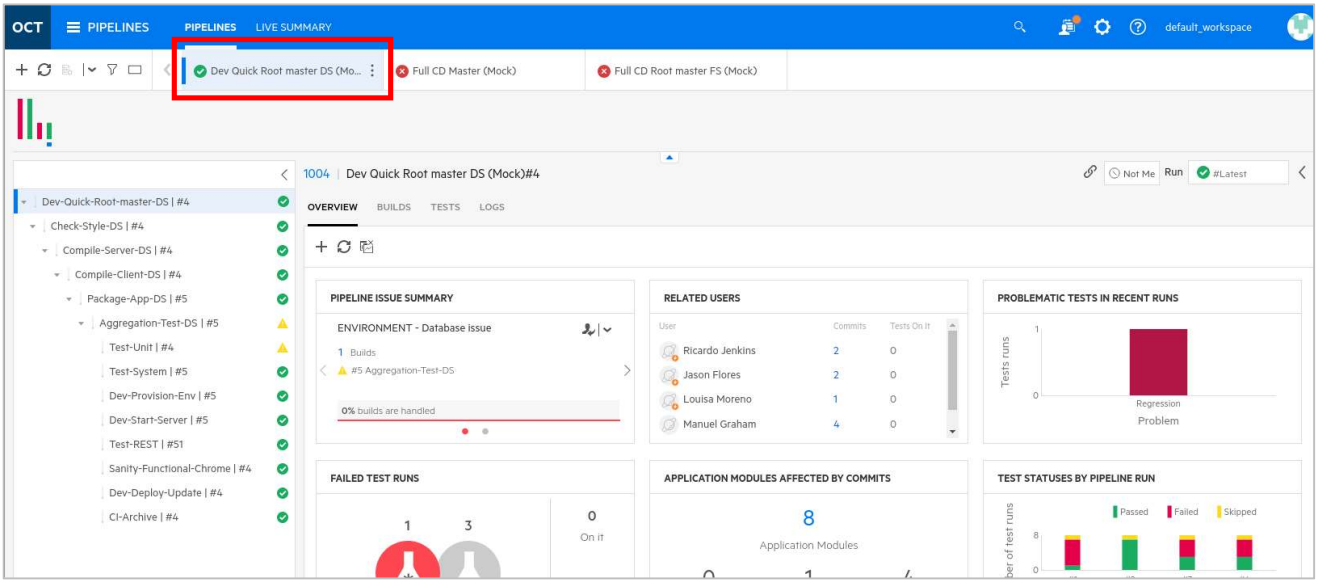
ALM Octane implements continuous quality in DevOps software delivery. ALM Octane is designed for fast-moving App Dev and test teams that are adopting continuous integration (CI) for DevOps. Test execution is intrinsically linked to CI and results are automatically available in ALM Octane's context-driven quality and defect management dashboards. Once test execution is completed, users can analyze test results, fix defects, as well as monitor both individual releases and overall product quality.

Testing and quality insights are reflected and related to the builds, pipelines, and application modules in Octane so that agile teams can continuously assess the state of quality in the sprint.

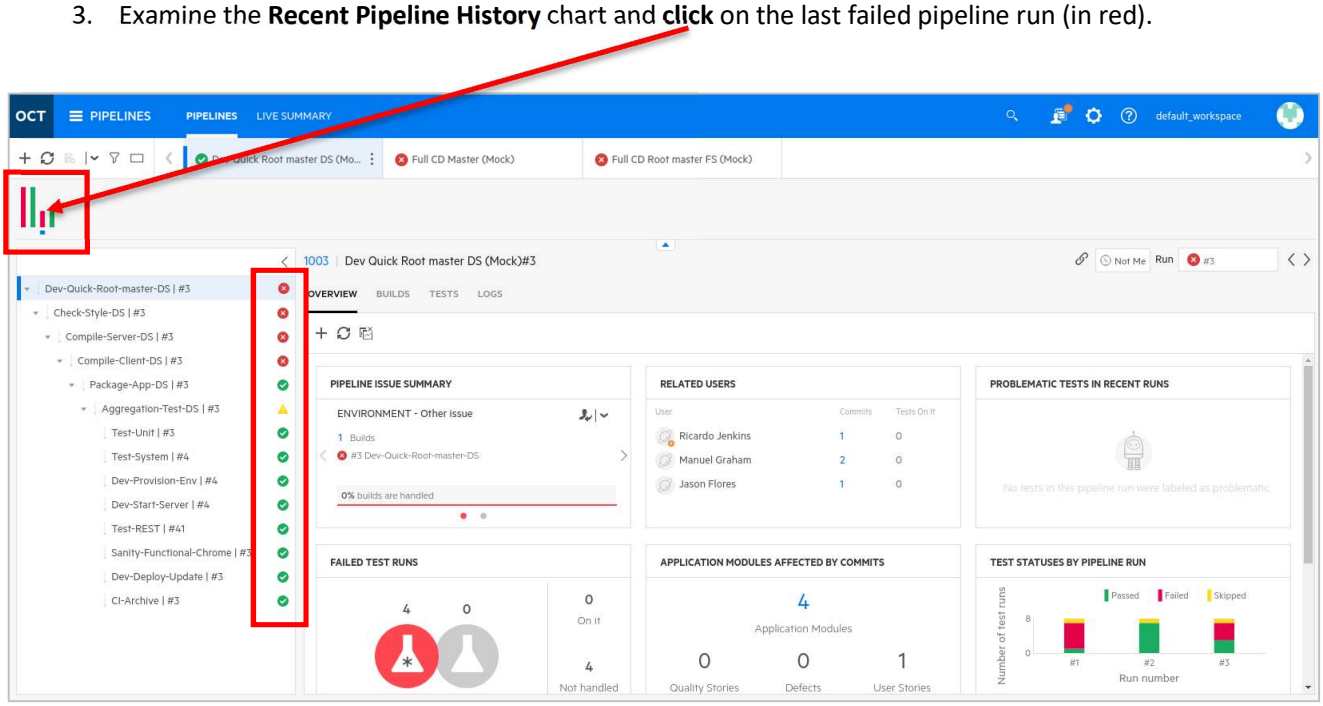
Reviewing Failure Analysis in a Pipeline

1. Navigate to the **Pipelines Module**. 

2. Select the **Dev Quick Root Master Pipeline** in the left side bar.



3. Examine the **Recent Pipeline History** chart and **click** on the last failed pipeline run (in red).



*** The top of the Failure Analysis tab displays several widgets that offer insight into your build and product quality. For example:*

- *<u>Failed Test Runs</u>: Displays the total number of tests that failed in this pipeline run. This number is broken down into newly failed runs and runs that failed previously as well. You can click on the number of New or still failing test runs to filter the Failed Test Runs tab accordingly.*
- *<u>Problematic Failed Tests</u>: Shows a breakdown of automated tests that have not been consistently successful, according to the type of problem. Click on the number of tests in a specific category to show only those tests in the Failed Test Runs tab.*
- *<u>Related Users</u>: Shows the people who need to be aware of the failures in this pipeline run. People whose commits were included in this pipeline run. An additional indication is provided if a commit seems to be specifically related to the failure. People assigned to investigate build or test run failures in this pipeline run. These are the users listed in the **Who's on it?** fields of the failures in this pipeline run. To dive into more detail click on someone in this widget to filter the list of failures. The lists displays only the failures where this person is in the **Who's on it?** field, or test run failures related to this person's commits.*

Understanding the Quality Module

In ALM Octane, you have the ability not only to monitor the progress of your work but also to track product quality across all areas of your application and project. Octane measures quality in a number of different ways, including test results, defects, or feature quality status. View this quality in the Application/Quality module and the Dashboard.

Application modules are the functional areas of the product. These areas are determined or structured according to different user processes, areas of the product, and so on.

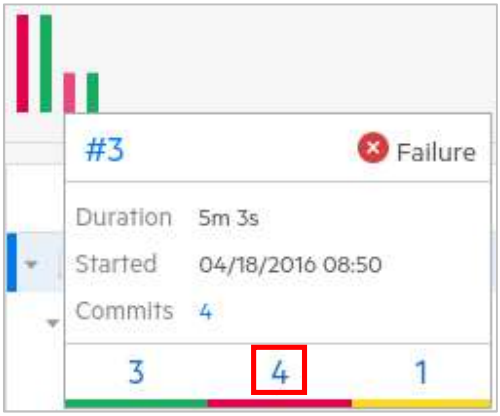
Using application modules is important, as in many cases, it is seen that many of the reported defects and testing problems that occur are limited to a small number of areas within an application. By setting up application modules, and associating tests, defects, and features, it helps you see the problematic areas of the application and concentrate testing efforts and development effort toward these areas.

Application modules are release-agnostic, which means that the details you view about the application module are global across all releases and reflect the overall health of the product.

Viewing the Quality Module

1. Make sure you're still viewing the **Pipeline** module.

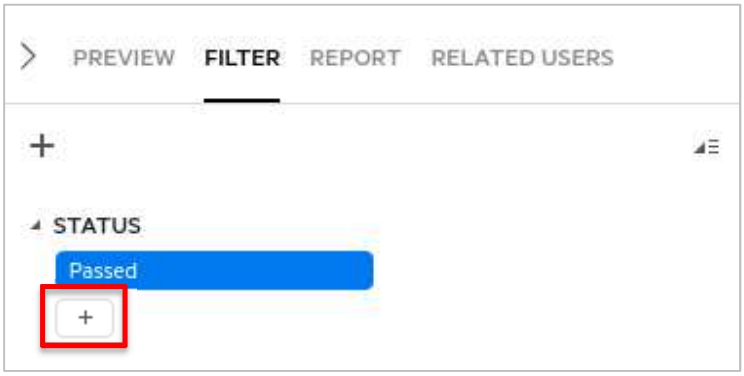
2. Hover over the pop-up status panel and select the number above the **red failed bar** and click the numeric link.



In the last build, **3** tests passed but **4** tests failed and **1** was skipped.

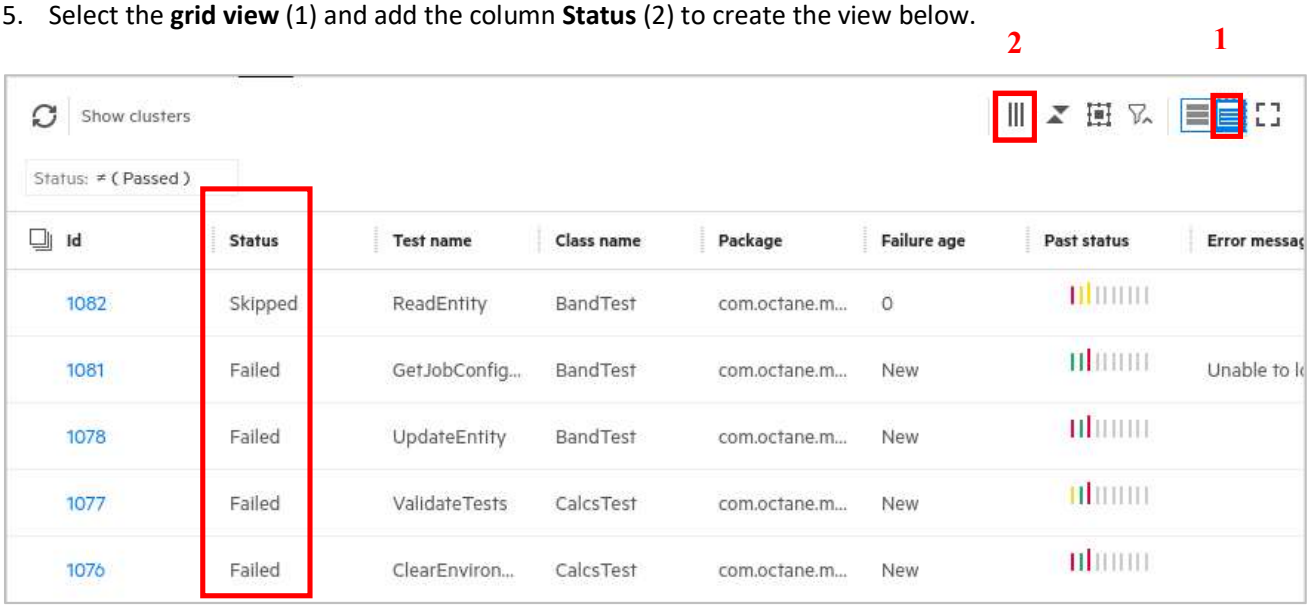
Click on the **Filters** icon to display which filter is being used and examine how this view was created. Try turning off the **Failed** filter to see all of the tests.

3. Another way to see just the **Passed** tests is to expand the rightmost section and select the **Filter** tab and then add the **Passed** filter as shown below.



4. Next, **right click** on the **Passed** text to **negate** the filter and show all tests that are not passed (failed and skipped). When the value is negated a line is drawn through the label.

5. Select the **grid view** (1) and add the column **Status** (2) to create the view below.



Good Progress!

You are learning how to manage overall Application Quality!

Exercise 8: Analyzing Data

Reviewing the Dashboard

The ALM Octane dashboard is the control center for analysis of your application's development and quality. It gives a visual, customizable display of how the application's development is progressing and what the level of quality is.

The dashboard is a collection of widgets designed to share data about your product development and progress. Out-of-the-box widgets include widgets to measure product quality within and across releases, progress within a release, test results, and other analytics. If the out-of-the-box widgets are not sufficient for your analysis needs, create a custom widget to display your data.

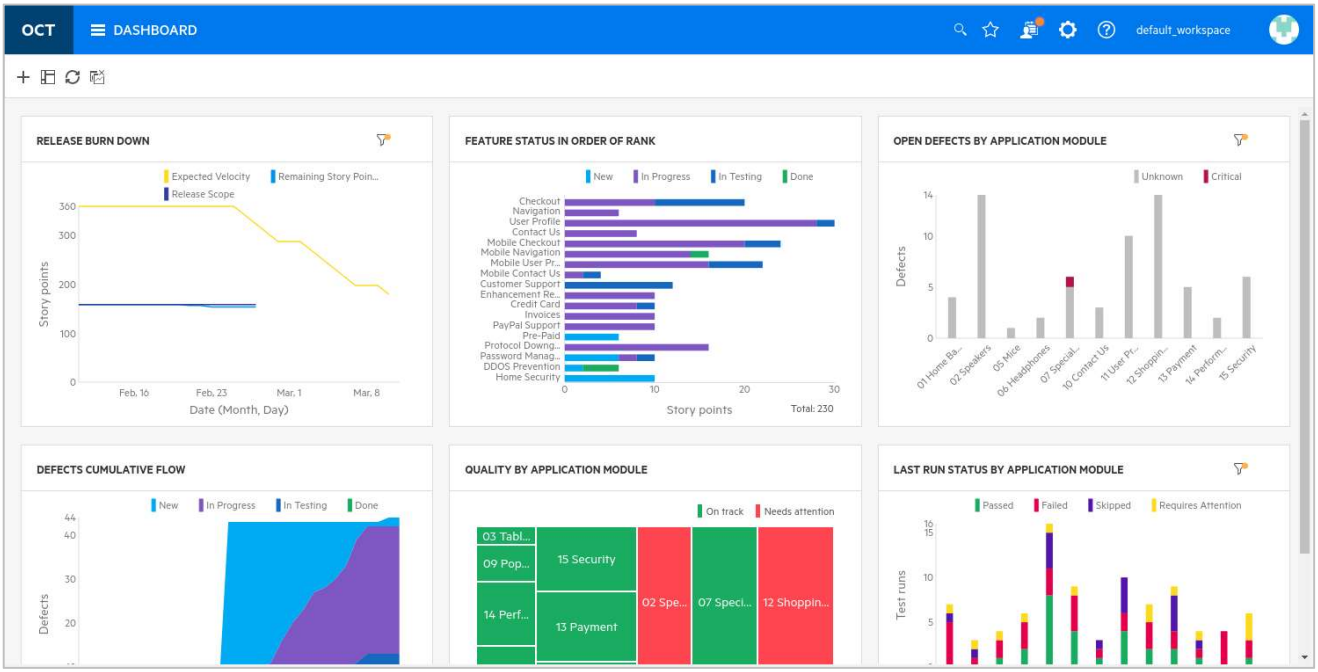
For the dashboard widgets, it is possible to customize the information shown in the dashboard such as the type of data reported, the scope and time frame, and how the data is grouped and displayed.

The dashboard also enables you to configure how you want to view widgets and save different widgets as favorites to use on repeated visits to the dashboard.

Dashboard Activities

The Main Dashboard is very much like the Overview Dashboards for individual Backlog items, except it covers all data points associated with the entire Workspace. Users are expected to leverage the data to visualize releases and overall product quality as well as to drill down on specific interest areas.

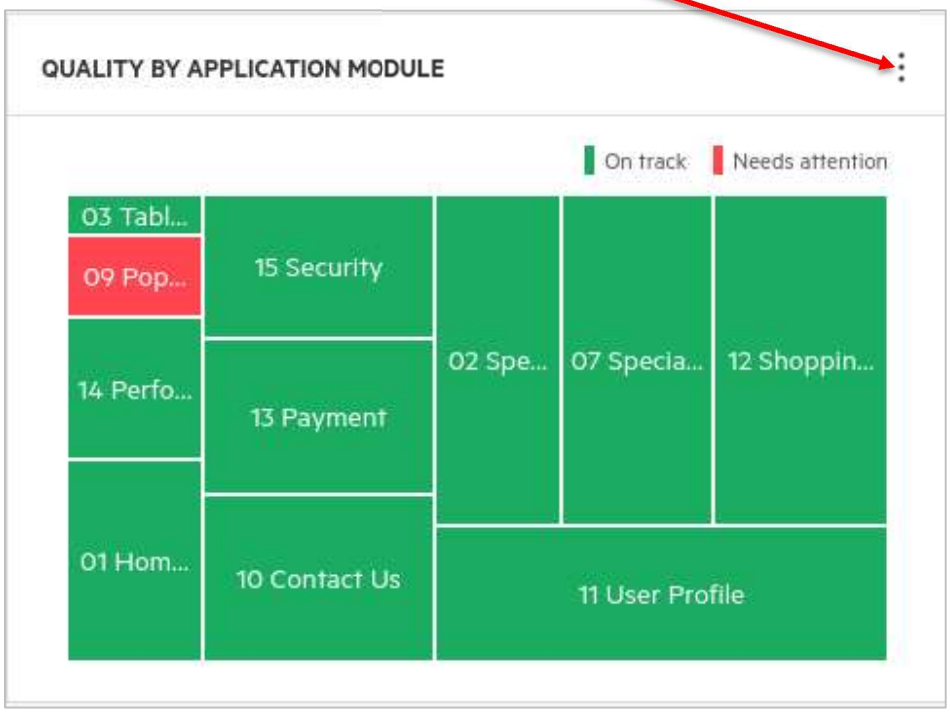
1. Navigate to the **Dashboard Module.** 



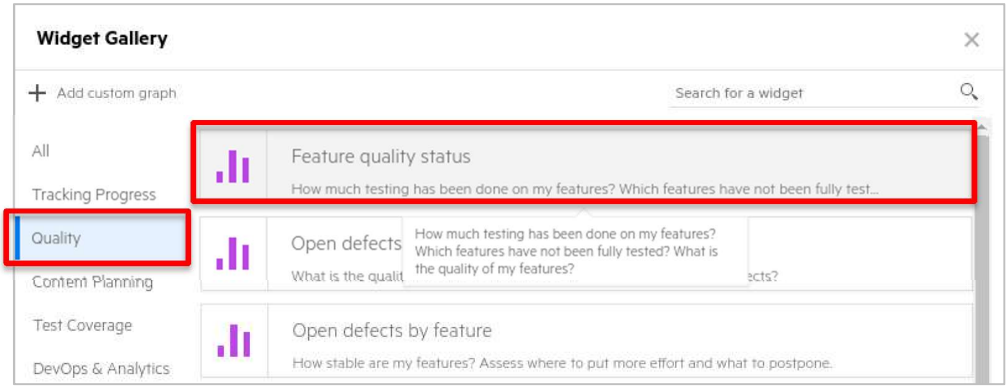
2. Select the **QUALITY BY APPLICATION MODULE** graph.

*** We are going to refer back to a graph that we viewed previously. The Quality by Application Module graph gives you a visual of the overall quality of your application. For the settings we have selected for this exercise this graph is showing you which parts of our application have test coverage.*

3. Configure this graph's **criteria** to show modules with **no test coverage**. It should show almost all green. That's good but could be all green if everything had test coverage.



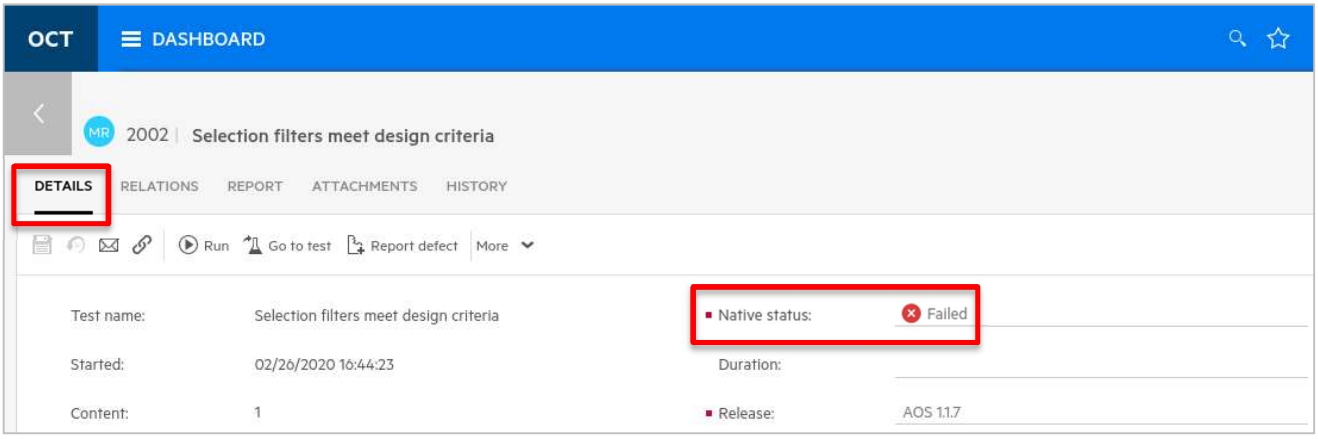
4. Create a new graph in the Dashboard module by clicking on the **+** symbol.
   From the Widget Gallery select **Quality** on the left and then the **Feature quality status** graph.



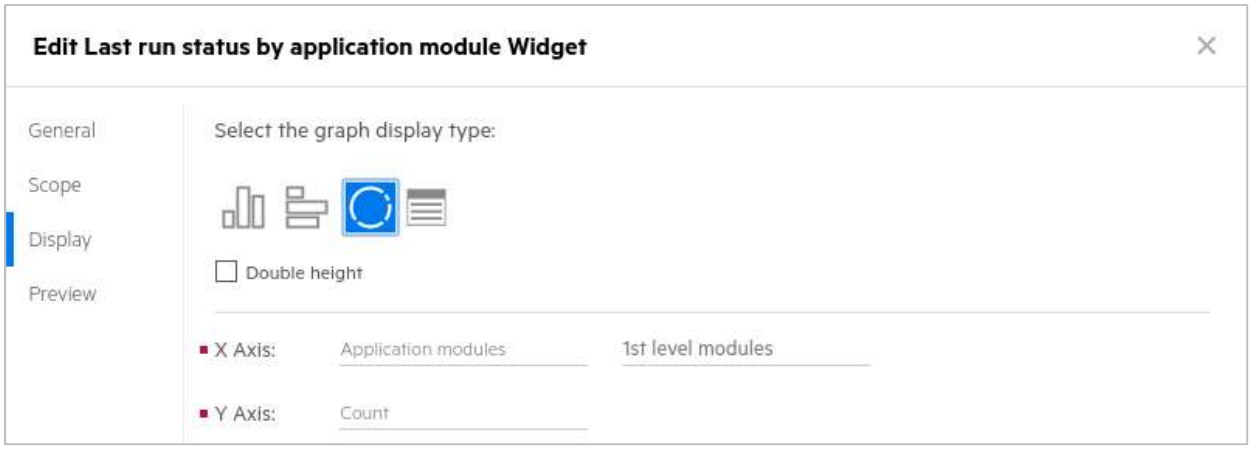
5. Select and expand to Full Screen the **FEATURE QUALITY STSTAUS** graph. In the legend in the upper right corner, select the **Passed** and **Blocked** options to deselect those filters. This will allow you to see only **Failed** and **Skipped** features containing Failed or Skipped Test Runs. It is easy to filter widgets to view information that is critical to you.

6. Click on the **Home Security** feature containing a **Failed Test Run** to drill down into the specific test run details. This is the manual test run we created previously.

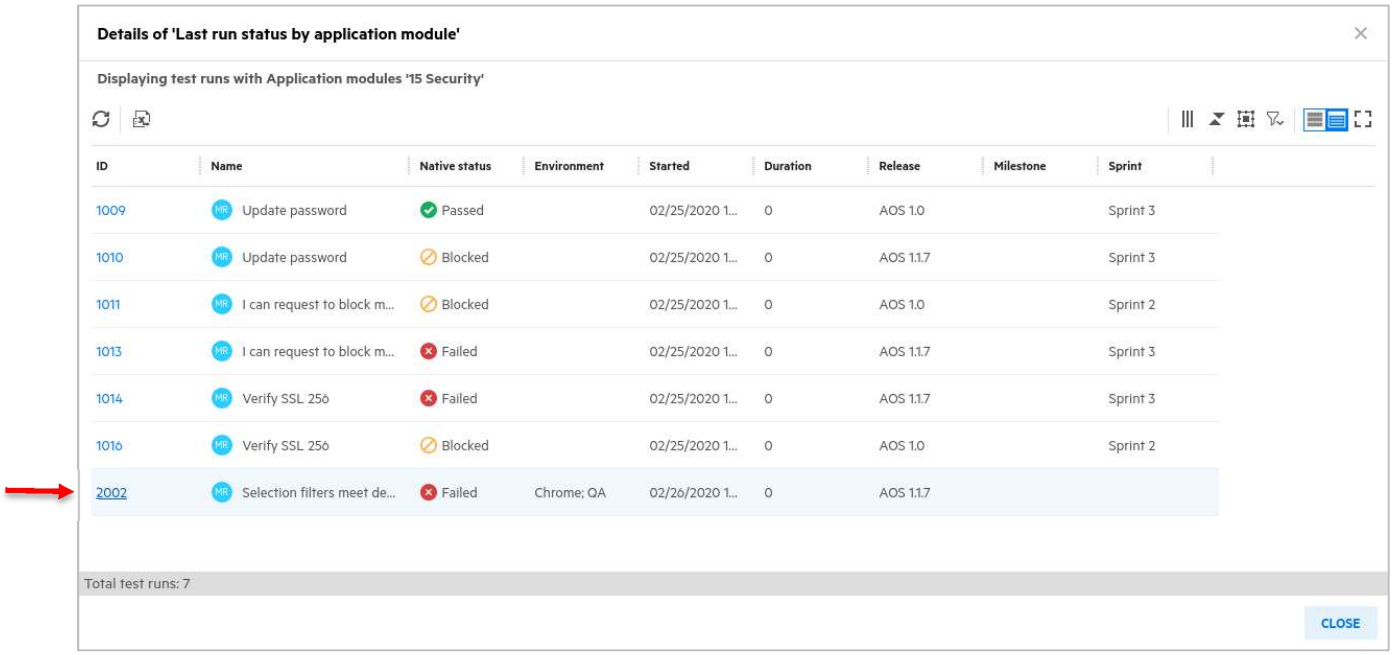
7. **Click** on the numeric link to see this test.
   Note that we can drill into a result and find this test with Native status: **Failed**.



8. **Click** the arrow to **go back** and **Close** the window and **Restore** the chart to the normal dashboard size.
9. Locate the existing graph widget "**Last run status by application module**" and configure it as a pie chart.



10. Mouse over the **15 Security** segment to view the test runs associated with that module.
    Click on that segment and scroll down to find your manual run (probably the last one in the list).

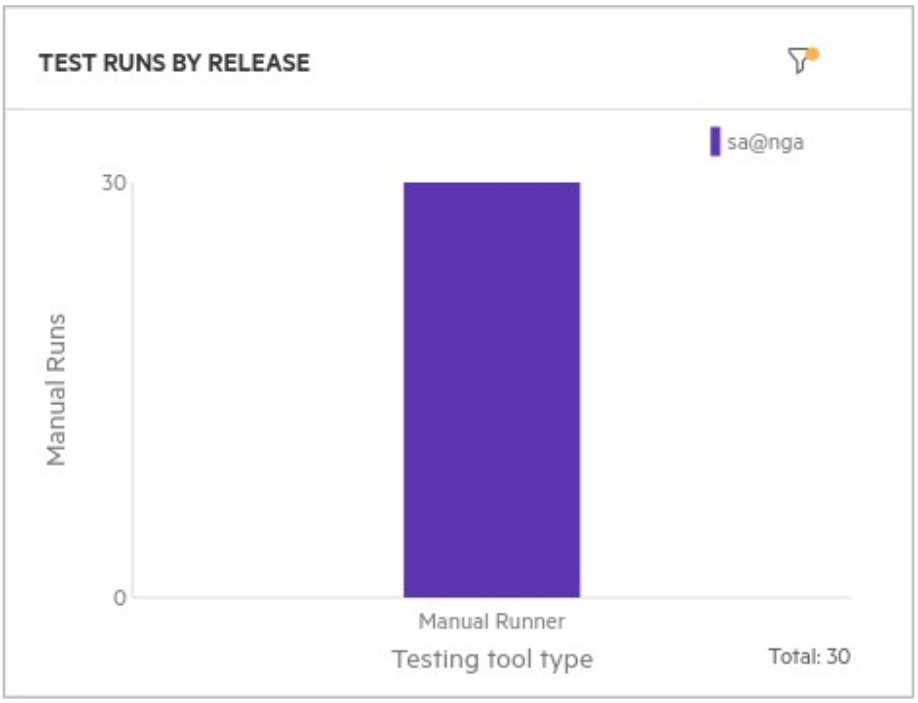


** *You can easily create custom widgets to monitor the quality and status of your application.*

1. Close the Details box and Create a **Custom Graph** by selecting the **+** icon and then select **Add custom graph** using the following instructions:

   a. Graph Name: **Test runs by release -> Next**
   b. Item Type: **Test run history (manual)**
   c. Release: **AOS 1.1.7**
   d. Run started in: **Last 7 days -> Next**
   e. X-Axis: **Testing tool type**
   f. Y-Axis: **Count**
   g. Group by: **Author -> Save**

2. Click on the colored bar to see the details of these tests.



*** This graph displays all tests run in the last seven days including the one that we created previously. Depending on when the container was created you may have more or fewer tests. You can also drill into the chart and find our previous test result and its defect.*

Wonderful!

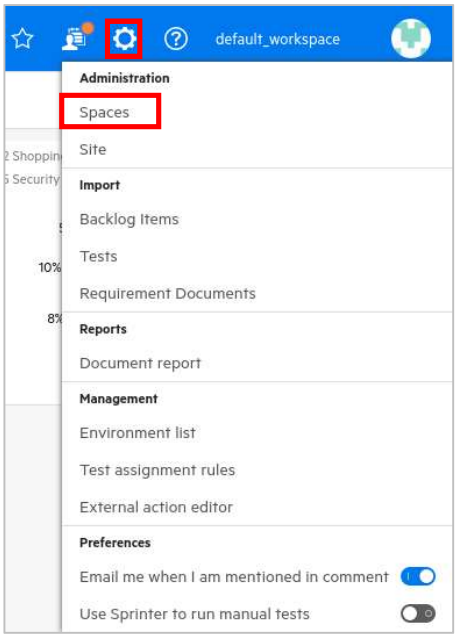
You have successfully analyzed the test results in the dashboard!

Exercise 9: Administration and Settings

Settings Overview

In this section we'll explore the configuration settings of ALM Octane

1. In the top banner, click the gear icon to access **Settings** and then select **Spaces**.



Data in ALM Octane is divided into separate areas within a larger environment. The separate areas are called workspaces. The environment is called the space. The spaces can be isolated or shared.
Workspaces associated with shared spaces can also share data.

Why do I need shared spaces?

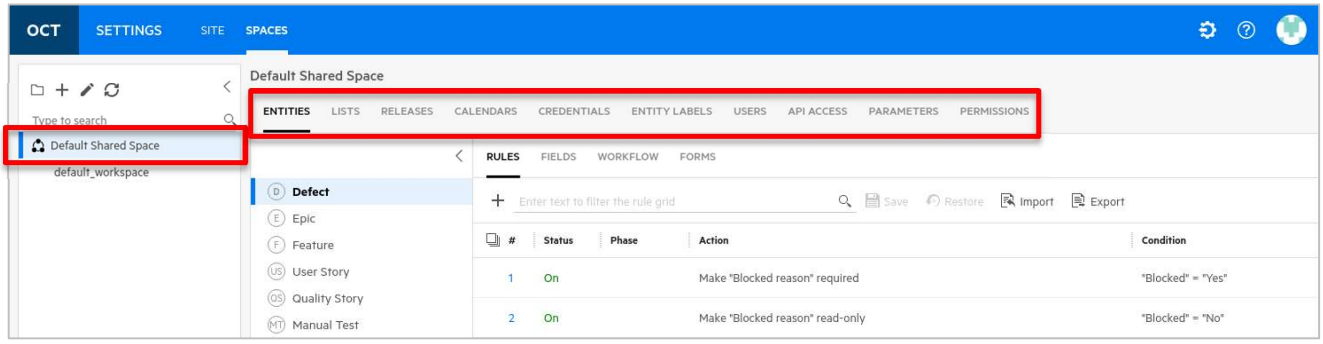
Often, application development projects might be similar but can share a core set of functionality. Imagine a global cellular company that wants to roll our services in North America and is expanding to Brazil. While much of the cellular application functionality remains the same, some pieces can be different with different country rules. For example, data retention might vary per local law. Consumers might also use their devices for different purposes to take advantage of faster (or slower) connection speeds. Billing and tax collection would also be different per country.

For these reasons you may want to compartmentalize these differences but also keep a core set of shared functionality. That's exactly what shared spaces are made for.

Shared space admins can create multiple workspaces to represent multiple projects, programs, or products managed on the same ALM Octane site. Shared space admins can also create users, and assign users different roles within workspaces. They also assign workspace admins for each workspace.

Workspace admins can manage data inside their workspaces that are associated with isolated spaces. This includes releases, teams, users, the backlog, application modules, tests, defects, and so on.
Workspace members can only access workspaces that they are assigned to.

1. Select the **Default Shared Space** on the left side list bar.

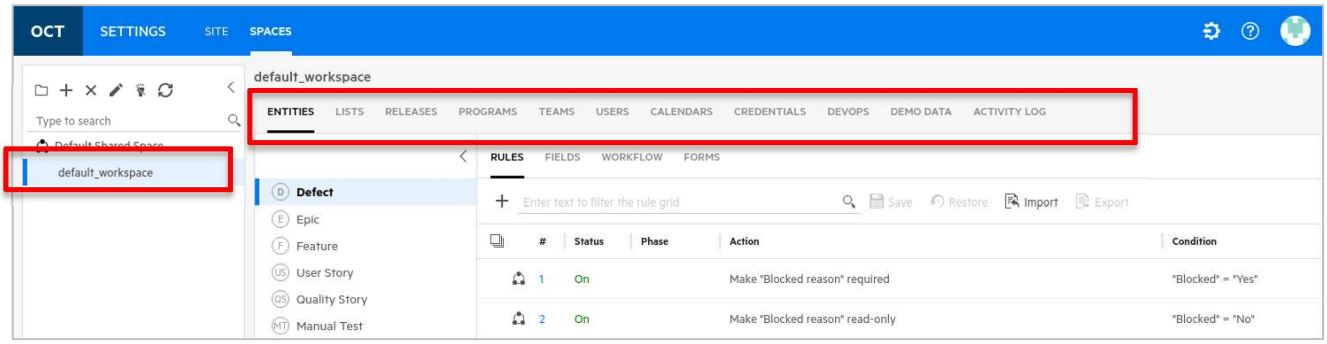


In this view you can see many of the same entities and options you have for the default workspace but changes made here are shared with other workspaces. Across the top of this view are various tabs for modifying Octane's shared space behavior.

2. Click on each of the section tabs to understand how Octane organizes its shared space data.
The details of these shared entitles is described below:

Shared Space Customization	
Entities	Applies to Shared Space - See **default_workspace** description in following table.
Lists	Applies to Shared Space - See **default_workspace** description in following table.
Releases	Applies to Shared Space - See **default_workspace** description in following table.
Calendars	Applies to Shared Space - See **default_workspace** description in following table.
Credentials	Applies to Shared Space - See **default_workspace** description in following table.
Entity Labels	Entity labels allow you to change (whitelist) the text ALM Octane uses to show various entities. You can even change the initials and plural forms used. In this way you can change the terms used for specific methodologies (e.g. SAFe, Waterfall, Spotify, etc.)
Users	Applies to Shared Space - See **default_workspace** description in following table.
API Access	Generates an API Key set (Client ID and Client Secret) for authenticating access to other integrated tools.
Parameters	A variety of adjustable parameters to control ALM Octanes behavior.
Permissions	Controls which roles have permissions to which modules/activities. Basically, who can do what?

3. Click on the **default_workspace** on the left menu and then click on each of these section tabs to understand how Octane organizes its data.



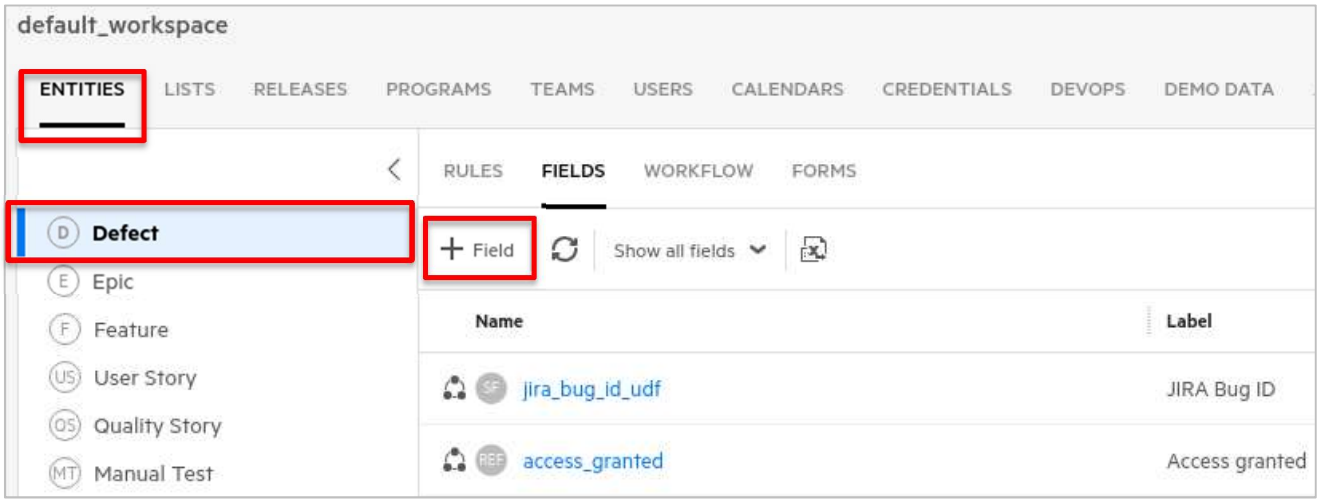
These sections are described below:

default_workspace Settings	
Entities	Entities can be modified here to create transition rules, user defined fields, workflow and form creation. Each Octane entity can be individually modified.
Lists	Lists are Octane items with defined values. For example Agile Type can be Kanban or Scrum. These lists are used throughout Octane and can generally be extended or new ones created. The behavior of these items is defined in this section.
Releases	Releases are the defined product releases, timelines and sprint count for the application being developed.
Programs	Workspaces can be organized into categories called programs. A program represents some kind of dev mission. For example, in a private banking workspace, you might create programs called DirectPay, MyAccount Mobile, MyAccount Web, and Services.
Teams	Teams are groups of individuals that work for a common purpose. Individuals can be members of more than one team.
Users	Users are the list of users defined in Octane along with their email and role they play.Users are typically also assigned to one or more teams.
Calendars	Public holidays along with religious, regional and personal calendars are defined here.
Credentials	Some applications require credentials. ALM Octane can supply these credentials when calling these applications from a Trigger Webhook rule.
Devops	The Devops section contains integration details for the more common Devops tools like CI, SCM, Security and definitions like Commit Patterns.
Demo Data	When evaluating ALM Octane customers may want to see what a mock workspace with connected data looks like. Fortunately, ALM Octane contains this mechanism to create a demo repository for this sort of examination. You can generate this mock data workspace from this section.

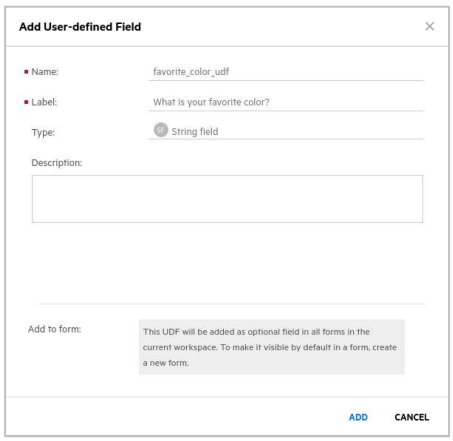
Adding User-Defined Fields (UDFs)

User defined fields or UDF's are a way to customize ALM Octane to fit an organization's needs. Many organizations have historical metrics or data that they want to associate with certain entity types. In Octane, you can create custom UDF's and even develop rules for how they work.

1. Select the **Entities** tab while in the **default_workspace**.
2. Select the **Defect** entity.
3. Select the **Fields** tab.



4. Click **+ Field** to add a new Field.
    a. Name: favorite_color_udf **NOTE:** Names must end in "_udf"
    b. Label: What is your favorite color?
    c. Type: String field
    d. Click **Add**

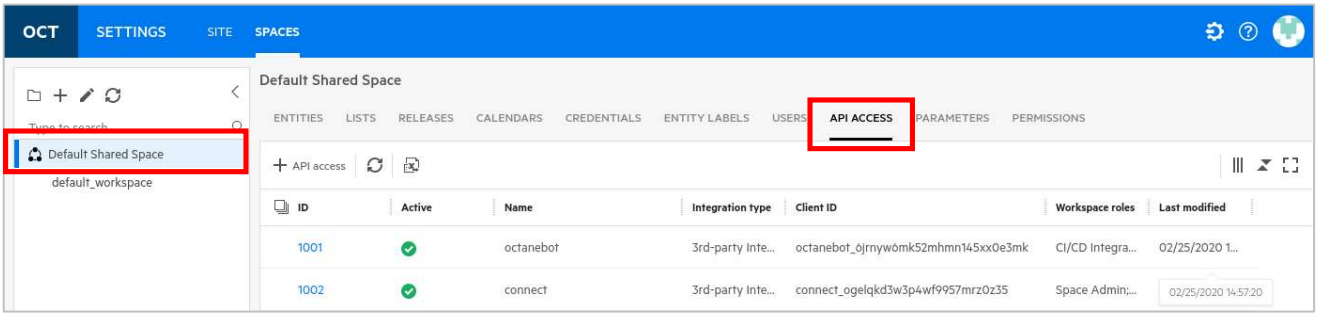


Once you have added this UDF, every new defect will now have this field available.

Creating API Keys

API Keys are encrypted credentials that can be used to access ALM Octane's data. We configured one previously to connect to Jenkins. In this section we'll create a similar key but we won't use it for anything.

1. Select the **Spaces** tab in the top left menu bar. Select **Default Shared Space** on the left hand side. Select the **API Access** tab.



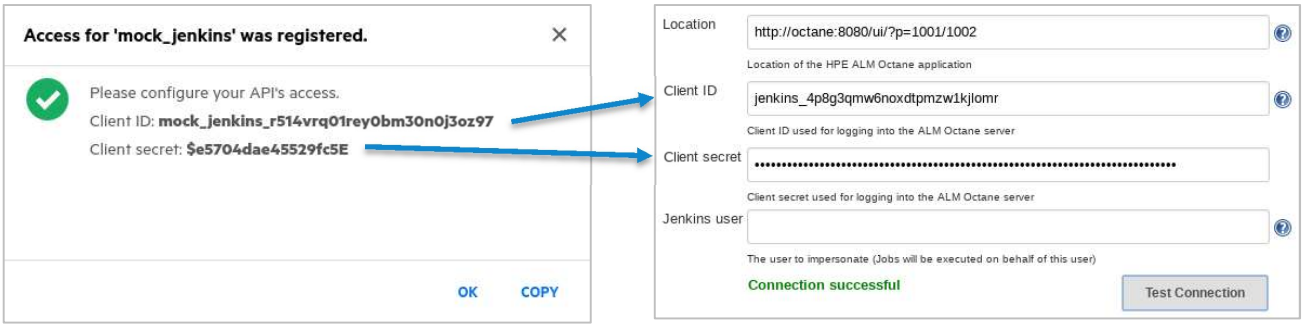
*** Set up API access once for each application that will communicate as a client with ALM Octane. This grants the application registered access keys to use for authentication. Two keys are created: Client ID and Client secret. For the exercises today we set up a second API key to integrate ALM Octane with Jenkins.*

2. Select + API access to create API keys.
   - Name: mock_jenkins
   - Role: CI/CD Integration
   - In Workspace: default_workspace

3. Click **Add**.

4. Write down or **Copy** the generated **Client ID** and **Client secret**. You won't see them again.

Once you have the **Client ID** and **Client Secret** you can use them to connect other tools like Jenkins.



5. Don't make any changes in Jenkins now – we were just showing you how the process works.

Creating Workflow Transitions

For many items (entities) in ALM Octane, you can define rules, workflow phases, lists, forms, and so on. One of the admin's early tasks is to make sure these items are defined according to the site's needs. Customizing items is essential to maximizing user productivity.

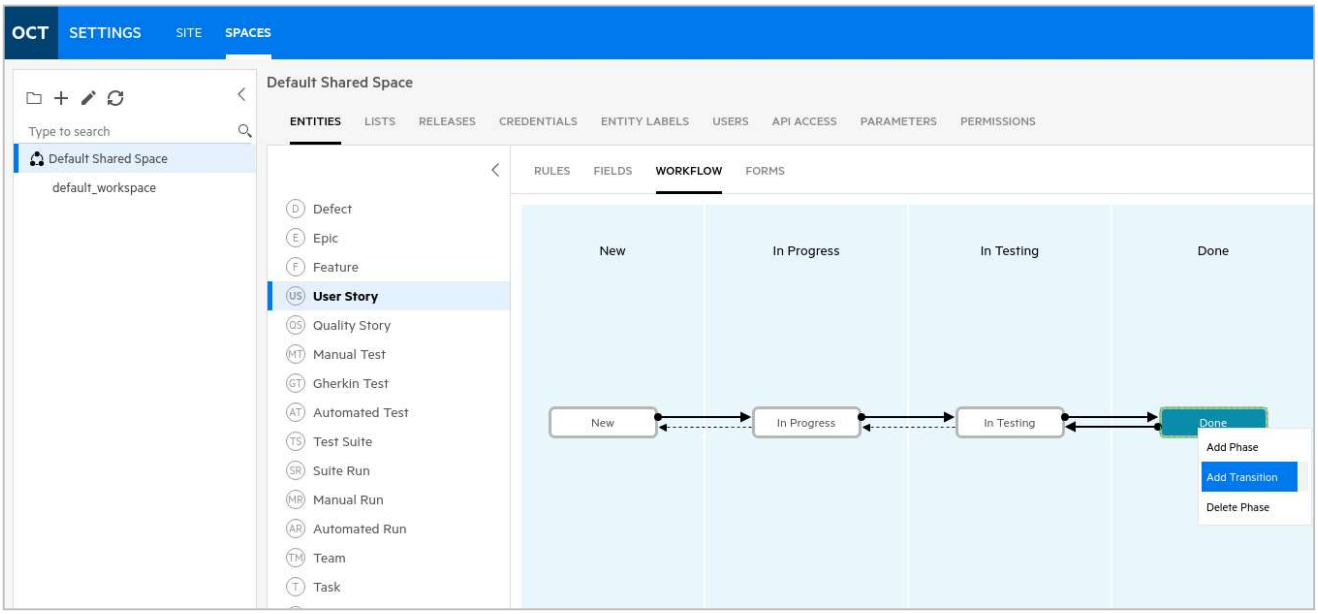
You can define the logical workflow for each item. The workflow determines what statuses the item can advance through as it is created, completed, and so on.

Workflows have metaphases, phases, and transitions. You can customize phases and transitions.

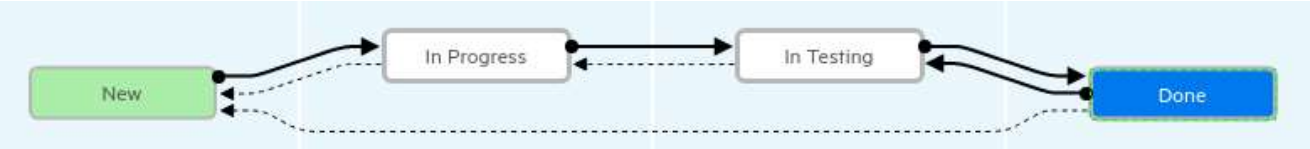
- **Metaphase:** The logical status of an item. Each metaphase contains one phase or more.
- **Phase**: The statuses of the item. A phase can be defined as the **Start phase**, which means it must be the first phase in the workflow.
- **Transition**: The permitted directions in which the flow can advance to another phase. You can set transitions as primary transitions, which indicate the main path for the flow.

1. Select the **Default Shared Space** on the left-side menu.
   The Default Shared Space is used since it has the most option to change the workflow.
2. Select the **Entities** tab.
3. Select the **User Story** item.
4. Click **Workflow**. The current workflow for the item displays.

Below is the default workflow for a user story. In this model, a user story cannot go from **New** to **Done**, without first being set to **In Progress** and then **In Testing**. This is the primary flow and these are shown as solid lines. Optional transitions are shown with dashed lines.



5. You can use the workflow editor to modify the workflow for the item.
6. Add a transition from **Done** back to **New** by right clicking **Done** to add a **Transition**.



7. Save your change by exiting the Settings section by clicking on  .
8. Using your newly acquired skills in Octane, locate a **user story** that is **Done** and see if you can now change it back to **New**. If so, HURRAY!

   Show your instructor what you've done!

9. Then go back and **delete** your transition (right click on it and select **Delete Transition**) to put things back the way they started. Nice Work!

Congratulations!

You have mastered the Administration and Settings section!

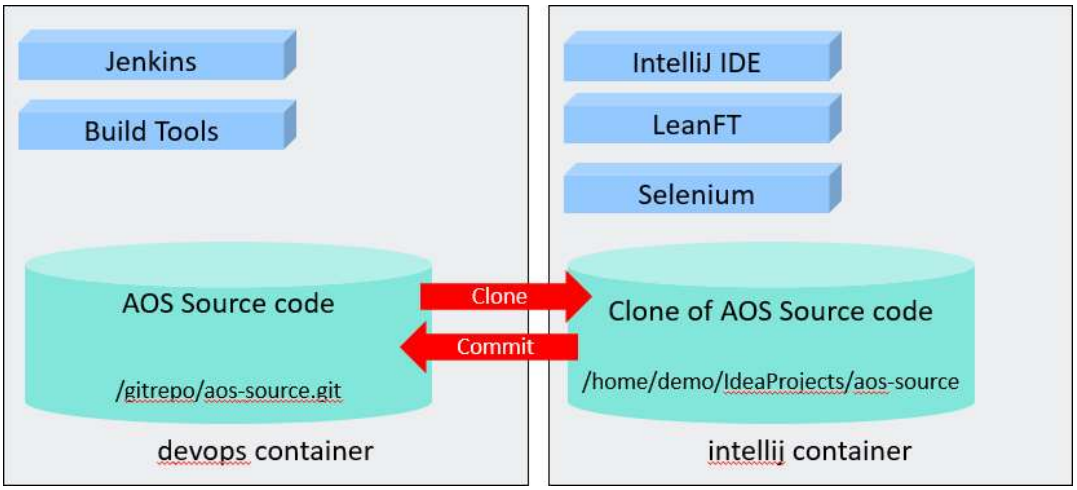
Exercise 10: Jenkins and DevOps Integrations

Jenkins is a freeware continuous integration (CI) system that manages and automates the building, deployment and testing of most modern applications. Because of its extensive library of plugins, Jenkins has become a multi-functional CI tool that can do almost everything in the DevSecOps arena.

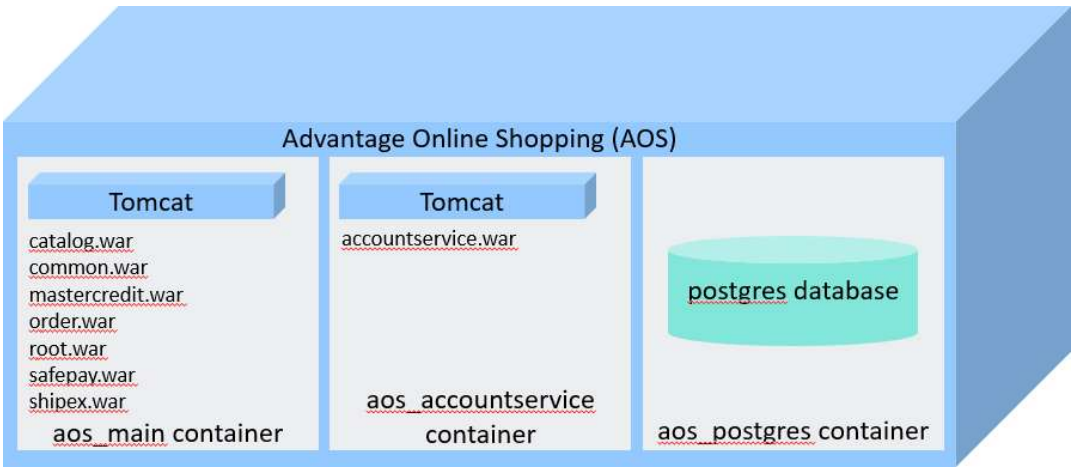
In this section well look at all the things that Jenkins can do along with how it integrates with ALM Octane.

The Nimbus DevOps Architecture

The containers available in Nimbus typically house an application, a webserver and a database. However, some containers, like devops or intelliJ, are more multifunctional and consist of several applications.
In the diagram below you can see how these two containers interact and clone/commit changes to each other.



In addition, the AOS (Advantage Online Shopping) application is divided into three containers.
As our e-commerce demo application, AOS was split up to represent a more typical docker architecture and also to allow for the ease of virtualizing the **aos_accountservice** using Micro Focus Service Virtualization.



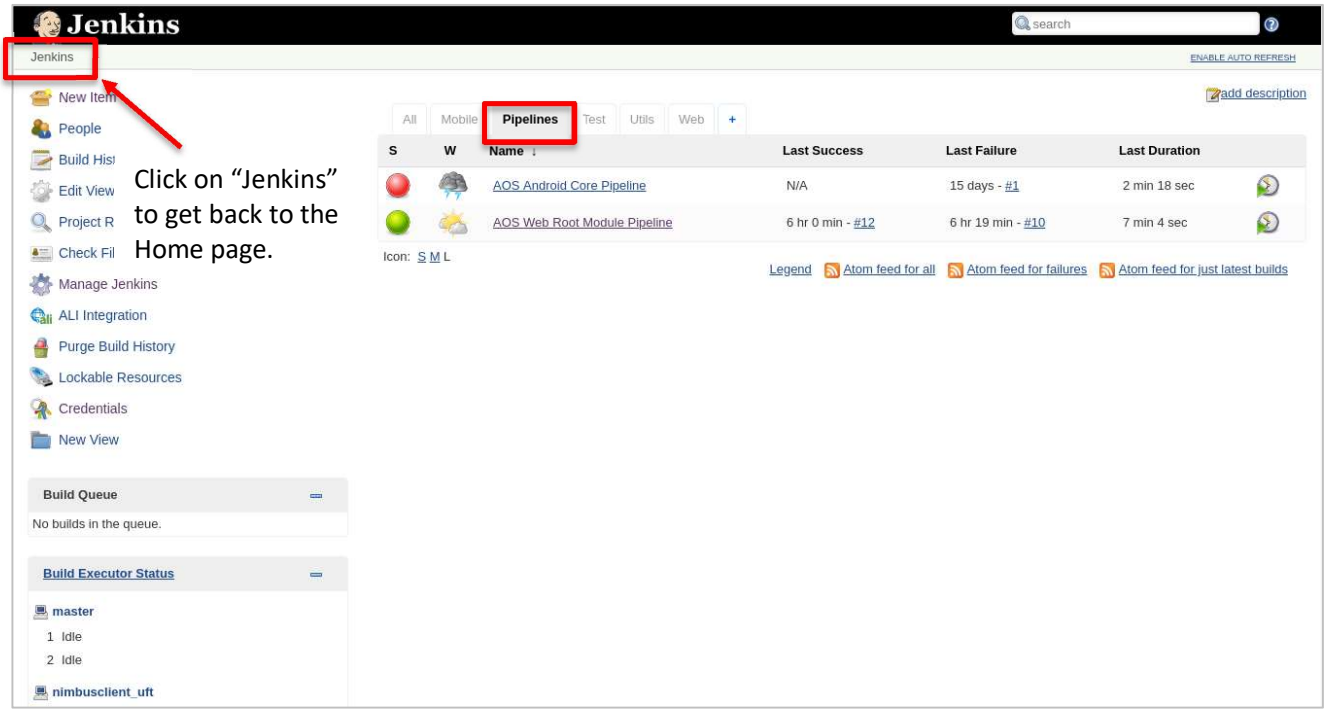
Getting Started with Jenkins

Jenkins at its core is a sophisticated task execution tool. You define jobs (or builds) in Jenkins and describe how to execute them. Jenkins, in return, keeps track of the execution with attention to the status of whether things passed or failed. With the help of "plugins" Jenkins connects with other tools to read build configurations like Maven's POM files and publish results to Fortify SSC or Octane.

1. From the NimbusServer VM, start (or verify) the **DevOps** container by typing:

   ```
   $ nimbusapp devops:2.2.1 up
   ```

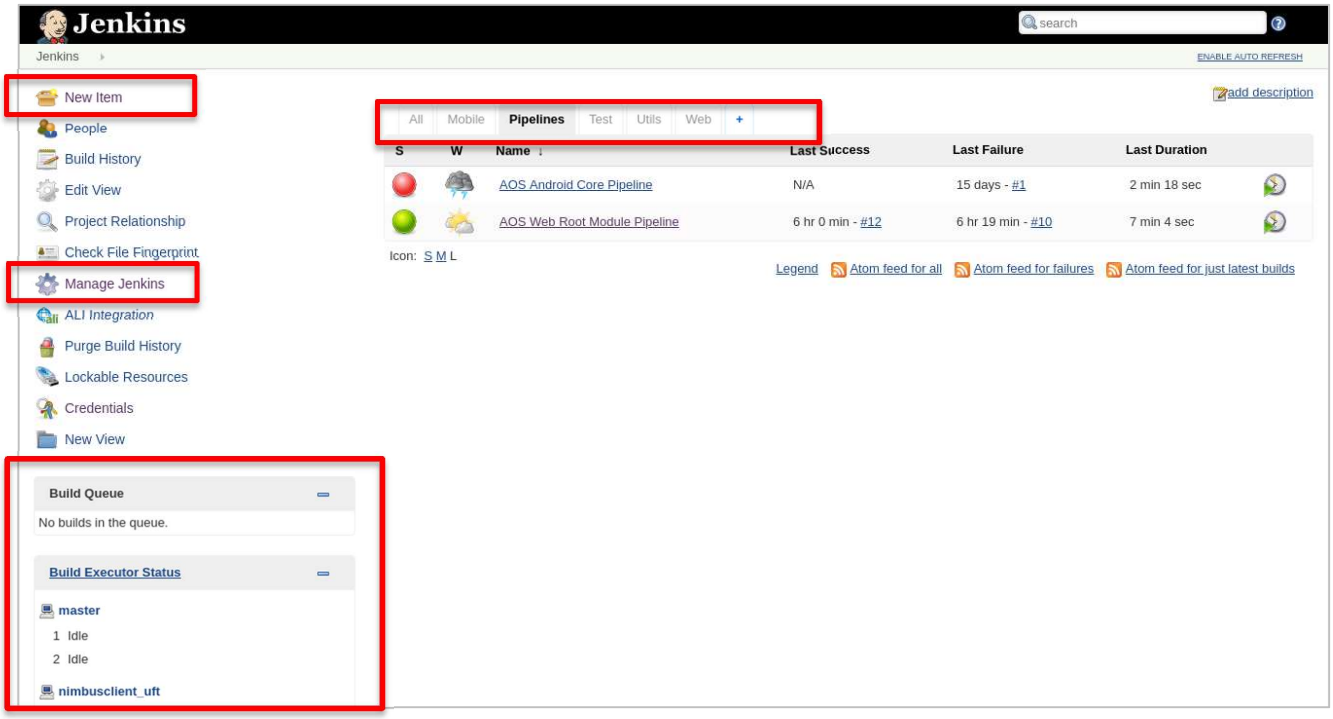
2. Wait a minute or so and then open the **Chrome** browser and select the Jenkins shortcut and the main **Pipelines** dashboard view as shown below.



The devops container has multiple software installations and purposes. In addition to housing Jenkins, it also serves as a git repository and can compile and build the AOS applications. It also houses the source code for Advantage Online Shoppins (AOS) in its local git repository.

The Jenkins Interface

The home page of Jenkins contains a list of jobs (also called builds or tasks) along with some links to configure Jenkins and to see if has any remote agents where Jenkins jobs can be run.



There are four areas here worth knowing about:

1. **New Item** – This link allows you to define new jobs and pipelines for Jenkins to run.
2. **View tabs** – This tabbed section shows your jobs and is filtered to show different types of jobs. If you have trouble finding a job by name, try the **All** tab since it always shows everything.
3. **Manage Jenkins** – This link allow you to configure Jenkins itself and install its plugins.
4. **Build Executor Status** – This section shows which remote execution agents are active. In this case, the NimbusClient VM has a remote agent that allows running tasks like UFT tests.

   These remote machines have agents running on them that communicate with Jenkins. Remote machines are also referred to as **nodes**.

Managing Jenkins

1. Click on the **Manage Jenkins** option on the Jenkins home page. 

2. Click on the **Configure System** option.
   You can ignore any notices about security items or reverse proxies – these are just notices.

   This section of Jenkins allow you to configure how Jenkins works and adjust settings for the installed plugins. Scroll down through the categories and we'll look at some of these options. For most of these sections, data has been pre-populated – you shouldn't need to edit them.

   a. **Maven Project Configuration**
      Apache Maven is a build automation tool used primarily for Java projects. There's usually a POM (Project Object Model) file that contains dependency and configuration details which are used when building the software.
   b. **SonarQube Servers**
      This third-party plugin gathers details on code metrics and vulnerabilities.
   c. **Micro Focus LoadRunner Enterprise Integration**
      This option produces additional debug information for LoadRunner Enterprise.
   d. **DA Server**
      This section contains the locations and parameters for MF Deployment Automation.
   e. **Fortify on Demand**
      This section contains the location of the demo FOD instance.
   f. **Application Lifecycle Management (ALM QC)**
      This section contains the name and server URL of the ALM QC container.
   g. **UFT Mobile**
      This section contains the name and server URL of the Mobile Center container.
   h. **ALM Octane CI**
      This section contains the name, server URL, Client ID and Secret and Jenkins user for the ALM Octane container. Since we're logging into Jenkins anonymously, we don't need to specify a Jenkins user. If your Octane container is running, you can click **Test Connection** and verify that Jenkins can communicate to Octane. Communications with ALM Octane goes through this plugin.
   i. **Service Virtualization**
      Communications to the SV Server is configured via this section.
   j. **LoadRunner Cloud**
      This section contains the connection details for LoadRunner Cloud.
   k. **Fortify CloudScan**
      This section contains the connection details for Fortify CloudScan.
   l. **Fortify Assessment**
      This section contains the details to connect to Fortify SSC (Software Security Center).
   m. **ALI Integration**
      This section contains the details for the ALI integration with the ALM QC container.

Managing Plugins

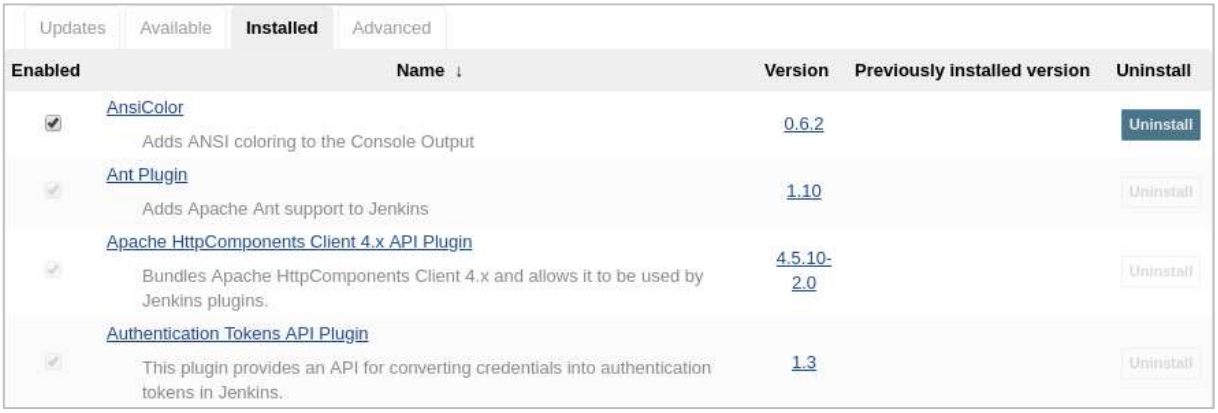
Part of Jenkins' popularity is its vast collection of publicly available plugins. A plugin is some additional functionality that can be added to Jenkins, like git integration. In this section we'll look at some of this useful functionality and examine some of the more popular devops plugins.

1. Click on the **Manage Jenkins** option on the Jenkins home page. 

2. Click on the **Manage Plugins** option.

3. The plugins section is divided into four tabs:
   - **Updates** – If any of your installed plugin have a publicly available update, they are shown here.
   - **Available** – If you want to search the list of publicly available updates, they are shown here.
   - **Installed** – The list of installed plugins on your Jenkins instance are shown here.
   - **Advanced** – Here you can set a proxy or upload a plugin file directly. Not all plugins are public.

4. Examine the list of **Installed** plugins. These plugins get updated very regularly and it's not uncommon to have many updates available (just like apps on your smart phone).



Note: Jenkins plugins have either a .jpi or .hpi extension. This stands for either Jenkins or Hudson Plug In. These files are actually just zip files and contain the directories and files of the plugin that will be installed into the Jenkins directory. These plugins all get renamed to have a .jpi extension and get placed in /usr/share/jenkins/ref/plugins/ inside the devops container.

5. Many plugins have a counterpart in the **Manage Jenkins** section but often these can have different names. For instance, the **DA Deploy** plugin corresponds to the **DA Server** section in **Manage Jenkins**.

6. Scroll down through the list of Micro Focus related plugins.
   How many can you find that are directly related to Micro Focus? _____

7. **Navigate** back to the Jenkins main dashboard by clicking on Jenkins (if prompted, select **Leave**).

The Octane - Jenkins Integration

The ALM Octane – Jenkins integration incorporates data from your Jenkins CI pipelines into your application delivery process, helping you analyze quality and progress.

You can find more information on the Octane - Jenkins integration here:
https://admhelp.microfocus.com/octane/en/latest/Online/Content/AdminGuide/jenkins-integration.htm

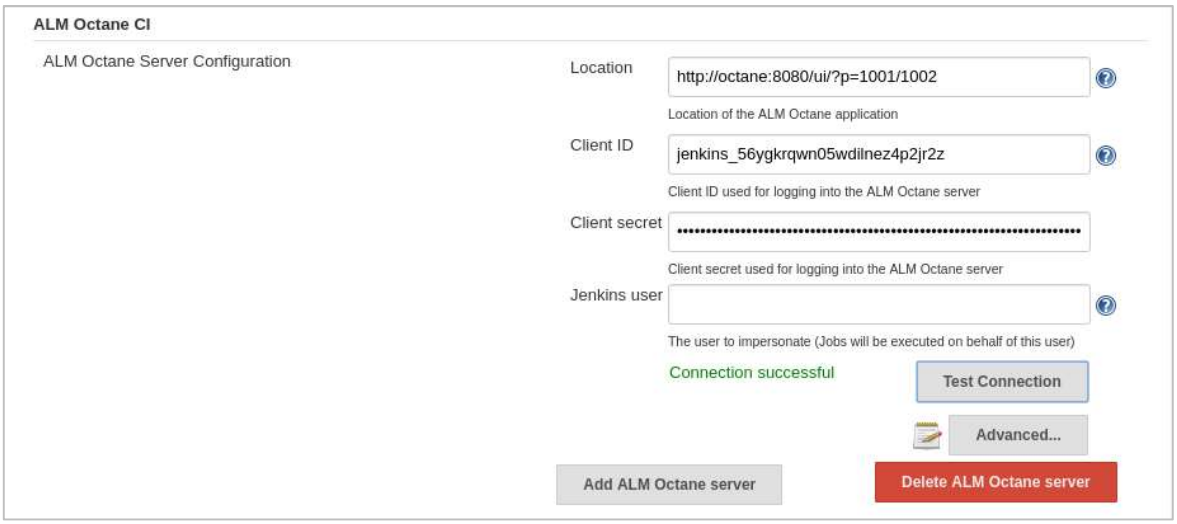
1. From the NimbusServer VM, start (or verify) the **DevOps** and **Octane** containers by typing:

```
$ nimbusapp devops:2.2.1 start

$ nimbusapp octane:15.0.46.68 start
```

1. Click on the **Manage Jenkins** option on the Jenkins home page. 

2. Click on the **Configure System** option.

3. Scroll down to the section called **ALM Octane CI**.



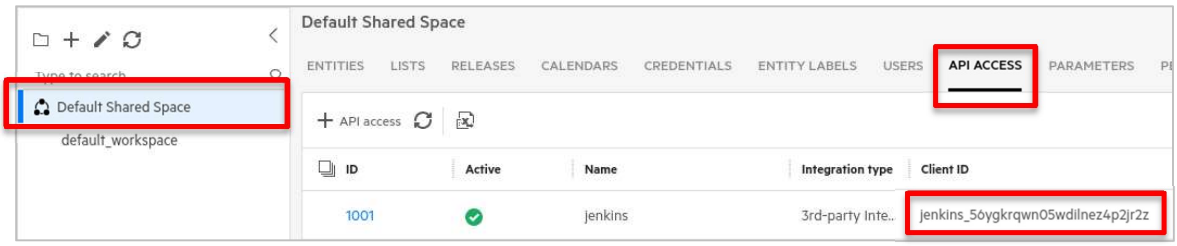
4. This section (which is pre-populated in our devops container) contains the connection details for Octane including:
   a. **Location**: The URL and shared space ID parameter
   b. **Client ID**: The unique client ID generated by Octane
   c. **Client secret**: The unique client secret generated by Octane
   d. **Jenkins user**: The user to be impersonated (blank in our case)

5. Open a new browser tab to the Octane shortcut and click **Login**.
   - **Name**: sa@nga
   - **Password**: Password1

6. Click on the gear (Settings) icon in the topmost tool bar and select **Spaces** under Administration.

7. On the left side select **Default Shared Space**.

8. In the tabs to the right select **API ACCESS**



This section of Octane's administration function is where you configure the integration with Jenkins.

Verify that the Jenkins Client ID here matches the one shown in the Jenkins **ALM Octane CI** section.

Running a Jenkins Pipeline

When Jenkins runs a typical job it creates a temporary place to house files and artifacts. This could be the intermediate files of a compilation or the output results of a test execution. The details on what happened during a job build can typically be found in the build's **workspace**.

A pipeline is a collection of jobs that run sequentially, but certain tasks can also run in parallel.
In our Jenkins instance we have defined two pipeline jobs that build:

1. The AOS Android Mobile app
2. The AOS Web application

Pipelines don't have their own workspace but the jobs they contain typically do.
These pipelines do more than just compile code – they perform the following types of tasks:

1. Create a Jenkins build workspace
2. Check out code from a git repository
3. Compile the AOS source code (using a Maven **P**roject **O**bject **M**odel file (POM file))
4. Scan the code for security vulnerabilities and code health
5. Run JUnit tests as part of the build process
6. Un-deploy and redeploy the AOS web modules
7. Run regression tests (LeanFT and Selenium) to validate the newly built application

In the following section we'll run a simple Pipeline job and examine what comes out of it.
Always wait a couple of minutes after each **nimbusapp** command to give it time to boot.

1. From the NimbusServer VM, start (or verify) the **DevOps**, **AOS, IntelliJ and Octane** containers:

   ```
   $ docker login      (Your instructor will provide the credentials)
   $ nimbusapp devops:2.2.1 start
   $ nimbusapp aos:2.2 start
   $ nimbusapp intellij:2.2.0 start
   $ nimbusapp octane:15.0.46.68 start
   ```

   NOTE: The IntelliJ container runs UFT Developer but it closes it down after4 hours of inactivity. If your IntelliJ was running for over four hours you must restart UFT Dev from the menu UFT Developer > **Disable UFT Developer** followed by UFT Developer > **Enable UFT Developer**

2. From the NimbusServer VM, open a browser to the **Jenkins** home page and select the **Pipelines** tab and **click** on the pipeline **AOS Web Root Module Pipeline**

3. Select **Build Now** (on the left border) and wait while the pipeline finishes.
   Click on the flashing ball icon and you can see the **console output** showing every step.

Near the final stage of this pipeline, it runs a test called **AOS Web Regression Test**
We'll examine this test to learn more about Workspaces.

4. Once the pipeline has completed, go back to the Jenkins home page and click on the **Tests** tab and then click on the test **AOS Web Regression Test**

5. From this view you have links to the full **Workspace** folder, the **runresults.html** file and to the **UFT Dev report** icon.

6. Click on the **runresults.html** link and view the last successful UFT Dev run results.
Note that if this test had failed, you wouldn't see that failed set of results here. You would need to drill down to the specific run (e.g. click on the #6) and look inside its workspace.

7. Click the **back** arrow on your browser to go to the previous page.

8. **Click** on the **Workspace** link.

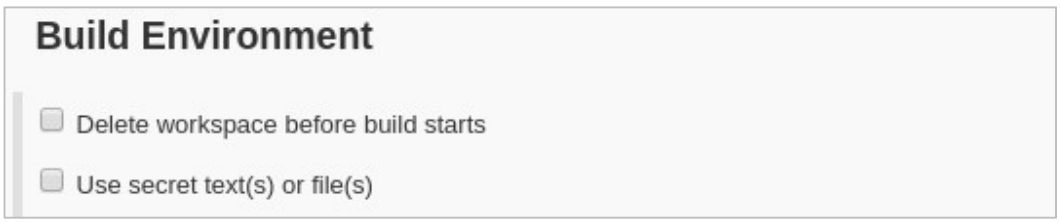
9. Here you can see the entire contents of the workspace.
   Click on the **RunResults** folder and then on then on the **runresults.html** file.
   This is the same results that you saw before (because this test was most recently passed).

   Go **back** to the previous page via your browser back button.

10. Click on the UFT Dev icon in the Build History section.

   This is a quick way to instantly navigate to the **LeanFT run results** for that build.
   Go **back** to the previous page via your browser back button.

11. Click the **Configure** link on the left side and scroll down to the **Build Environment** section.



12. Note that we've not checked the option to "**Delete workspace before build starts**".
    This would remove all old artifacts (like UFT Dev Run Results) before it starts a new run.

13. Scroll down to the **Post-build Actions** section.
    This option will save all the specified files to the job's workspace.

For UFT One or UFT Dev tests it makes sense to have this enabled since you can quickly navigate to the results of any specific test run.

14. From the AOS Web Regression Test, click the **Latest Test Result** link.

15. Continue to drill down on the links ((root) > LeanFTTest) until you find the status of the five tests that were run. Note that one of the tests explicitly checks the label of the Speakers category.

16. Click on the **Jenkins** breadcrumb to go back to the Jenkins home page.

Exercise 11: Tracking Code Changes in Octane

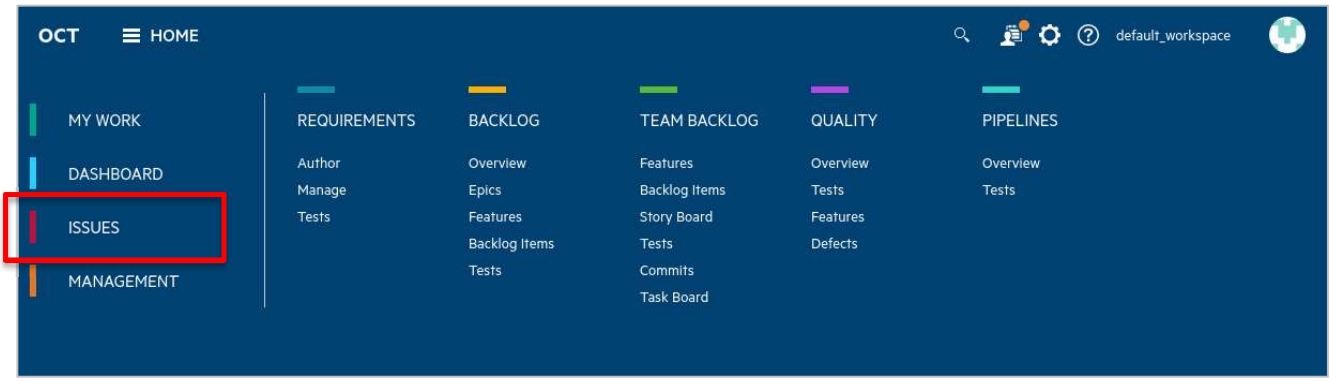
Octane and IntelliJ communicate by means of the Octane integration and also the IntelliJ plugin. With these mechanisms in place we can link source code commit changes to octane backlog items. This process also makes for a great DevOps demo since we create a defect, assign it to a developer, fix the defect and push the change and then build a new application along with testing.

1. First we need to start the **IntelliJ** container which holds the IntelliJ IDE and the UFT Dev agent.
   From the NimbusServer VM, start (or verify) the following set of containers by typing:

```
$ nimbusapp devops:2.2.1 start
$ nimbusapp aos:2.2 start
$ nimbusapp octane:15.0.46.68 start
$ nimbusapp sca:19.1.2 start
$ nimbusapp ssc:19.1.0 start
$ nimbusapp sonarqube:7.7 start
```

2. Open a browser and navigate to the Octane browser shortcut.
   Username: sa@nga
   Password: Password1

3. Click the Octane **HOME** button and select **ISSUES**.
   We're going to create a defect and then associate that defect to code that we'll change later.



4. Create a new defect by clicking on the + Defect sign and enter the following fields:
   Name: Need to change Speakers label to Audio to cover more device types.
   Severity: Medium
   Application Modules: 02 Speakers

   Click **ADD & EDIT**

5. Set the **Owner** of the defect to be **sa@nga** as shown below:

6. Click the save icon to save your changes.

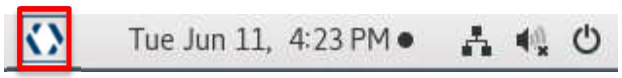
We've now created a new defect and assign sa@nga to work on it.
Next we'll go to our IntelliJ IDE and change the code to fix this defect.

7. From your terminal, type the following to open up the IntelliJJ IDE.

```
$ nimbusapp intellij:2.2.0 start
```

8. In a few seconds you should see the IntelliJ splash page appear (you may need to scroll to the bottom and agree to the terms and conditions). The UFT Dev agent starts automatically when IntelliJ starts.

9. You may also note that the UFT Dev agent has been started (look for a small icon in the upper right corner of your screen). This can be used to verify that UFT Dev is ready to execute tests.



Examining the DevOps File System

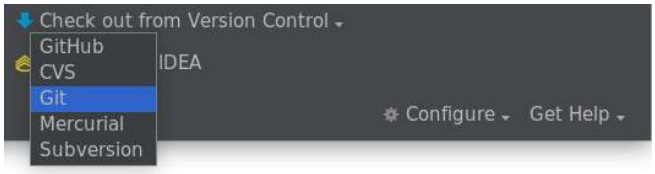
Since both the DevOps and IntelliJ containers can each have source code on them, we'll investigate the directories where that code lives.

1. Type the following command into a terminal window. This logs you into the devops container.
   ```
   $ docker exec -it devops bash    (or $ de devops)
   ```

2. We've now logged into the devops container which is where our Git-based projects live.
   Type the following commands:
   ```
   # cd /gitrepo
   # ls -al
   ```

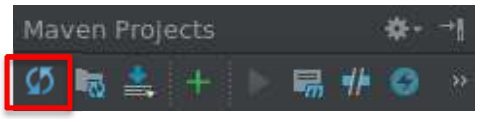
   You should see a list of IntelliJ project directories located in the gitrepo (/gitrepo) directory.
   It's these projects that we'll clone into the IntelliJ container and work with.

3. From the **IntelliJ** splash page **click** "**Check out from Version Control"** and select **Git**.



4. **Select** the **first** option, **aos-regression-test-web** and click **Clone**:

   In the popup **Checkout From Version Control** click **Yes** to open the project.
   When a Maven popup appears, select **Enable Auto-Import**
   In the **Maven Projects** panel at the upper right, select the **Reimport All Maven Projects** button.



   Wait for the project to load fully (a minute or two) and then select **View-> Tool Windows->Project.**
   You can also do this by typing **ALT+1**. This view allows you to navigate the files contained in this project.

5. **Go back** to your terminal window and type the following commands:
   ```
   # exit
   $ docker exec -it intellij bash    (or $ de intellij)
   $ ls -al ~/IdeaProjects
   ```

6. This final command will list the cloned projects contained in IntelliJ. You should see a directory called:
   ```
   AOS_Regression_Test_Web/
   ```

This is the IntelliJ project that we just cloned. The other item of note is the **~** (tilde) character. In Linux this denotes your home account's directory. So, the full directory path is

```
/home/demo/IdeaProjects
```

7. From IntelliJ, select **File->Close Project** to close the project but keep IntelliJ running.
   In your terminal window type the following to exit the IntelliJ container.
   ```
   # exit
   ```

Modifying the AOS Source Code

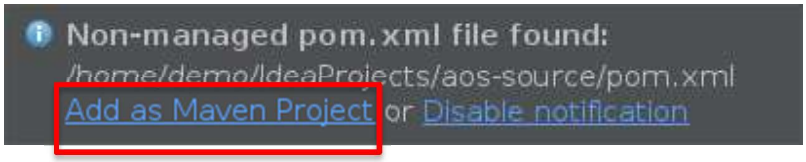
1. From the IntelliJ splash page **click** "**Check out from Version Control"** and select **Git**.

2. This time, **select** the **second** option in the pull-down - **aos-source** - and click **Clone**:

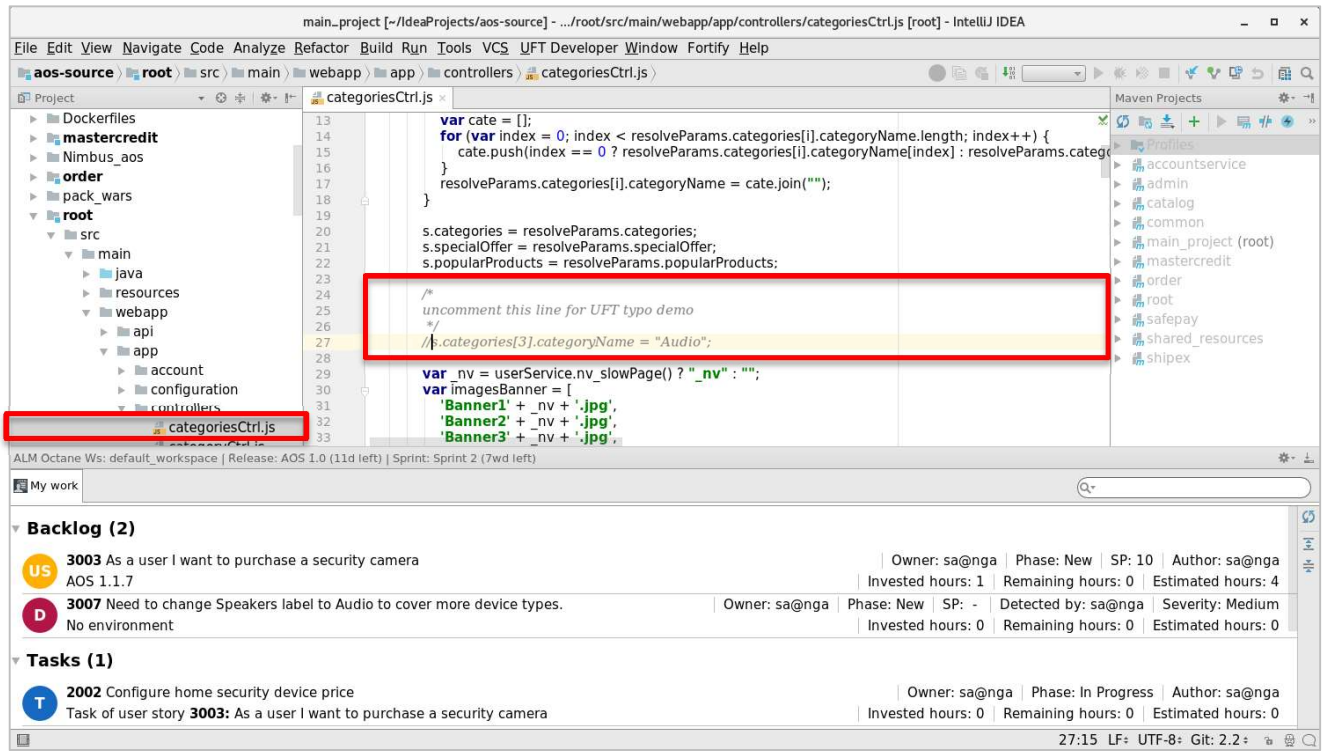
   This project contains the full source code for the Advantage Online Shopping mock ecommerce site.

3. When prompted, click **Yes** and choose any additional **default** options and then click **Finish**.
   Wait as IntelliJ loads this project and indexes the files. This can take a few minutes.

4. When it appears, click the **Add as Maven Project** in the lower popup (or click the **Reimport** button).



5. Type **ALT+1** to show the **Project** tree on the left. It can take several minutes to load completely.



6. At the bottom of the editor, verify that you are properly connected to Octane. You should see the **Backlog Defect** you just created and assigned to sa@nga.

   If not, go to Settings and configure the connection with the following data.

Server URL:	http://nimbusserver.aos.com:8085/ui/?p=1001/1002
Space:	1001
Workspace:	1002
Username:	sa@nga
Password:	Password1

Click **Test Connection** and then click **OK**.

7. Open the following path in the project tree on the left by clicking on the expansion triangles as shown:

   root > src > main > webapp > app > controllers

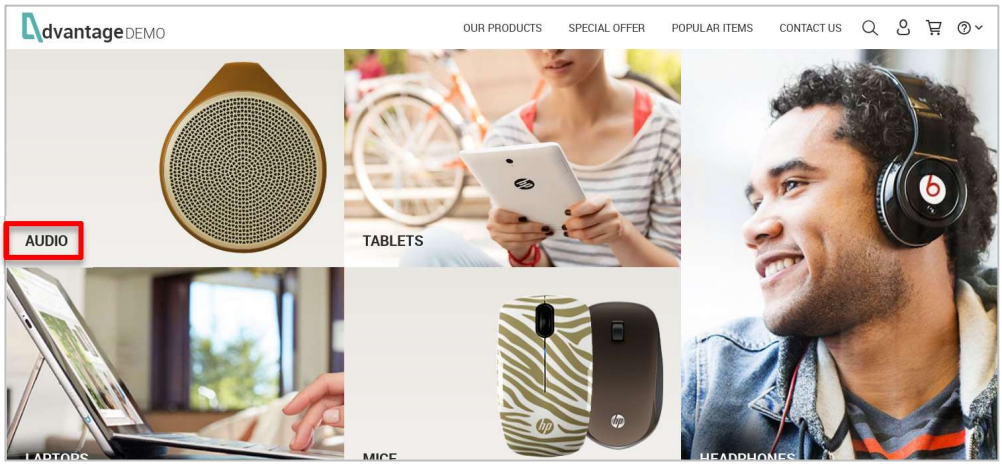
8. In the controllers section, double click on the **categoriesCtrl.js** file to open it.

9. In the code view in the middle, scroll down to **line 27**.
   This line contains code that can change the value of the **Speakers** category in AOS to **Audio**.

The DevOps Story

As part of our DevOps story we consider that a change request came in from marketing that the **Speakers** category should be changed to **Audio** to reflect a broader class of devices. Development takes this enhancement request and edits the code on line 27 and then commits the change and pushes it back to the Git source. Then, a Jenkins pipeline job is run to rebuild the AOS application and deploy it for testing. A regression test is run as part of that Jenkins build to verify that everything works as expected.

If you change this value to something like **Audio** and rebuild the application it will be reflected in the Main AOS home page as shown below. However, this will also cause a failure in the regression test (to be expected) that would need to be adjusted.



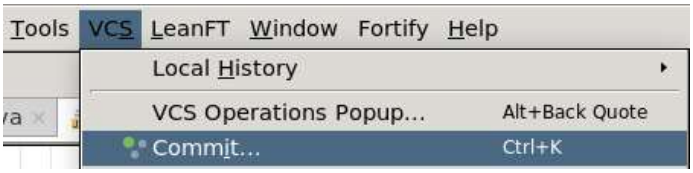
10. **Remove** the **two** leading slash comment characters (//) so the line reads as:

```
s.categories[3].categoryName = "Audio"
```

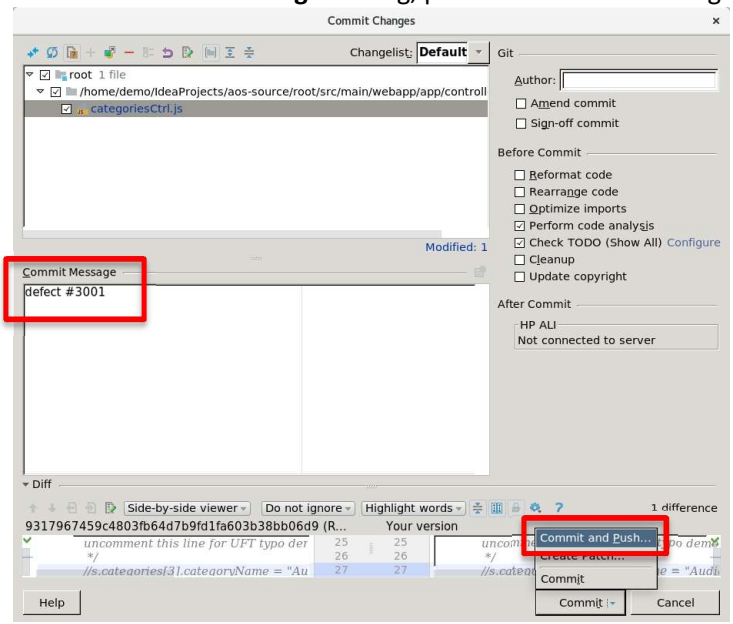
11. In the Octane integration section, right click on the defect and select **Copy Commit Message**. This tags the commit of code to the defect we created in Octane.

    You could also have selected Start work / End code which times how long it take to fix the code.

12. From the IntelliJ VCS menu, select **Commit…**



13. In the **Commit Changes** dialog, paste the commit message and **Commit and Push** the code change.



14. In the **Push Commits** dialog select **Push**

We've now done the following:

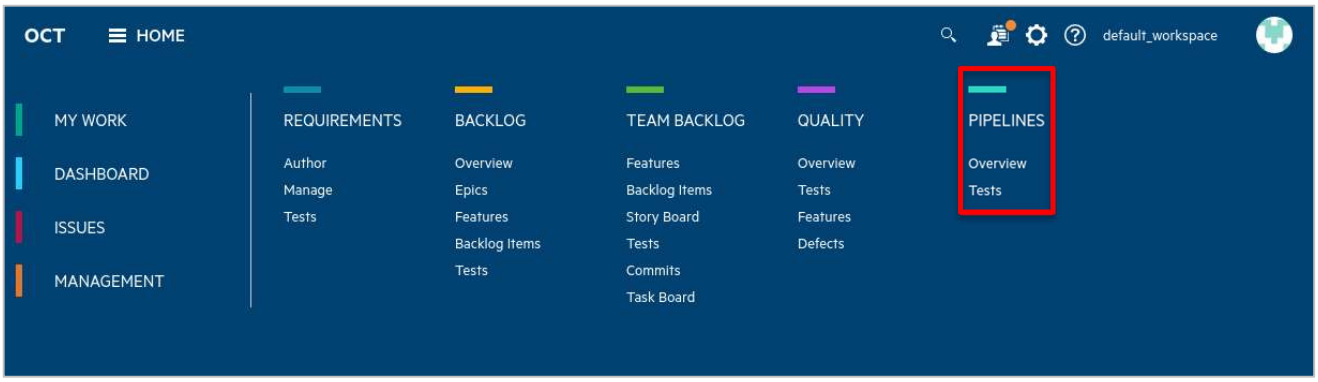
1. Created a new defect in Octane.
2. Assigned that defect to sa@nga.
3. Opened the AOS Source code in the IntelliJ IDE.
4. Retrieved the defect commit code from Octane.
5. Tagged our git commit with the commit code from Octane.
6. Committed our code back to the git repository on the devops container.

Next we'll need to rebuild our AOS application to include our change.

Running the Pipeline from Octane

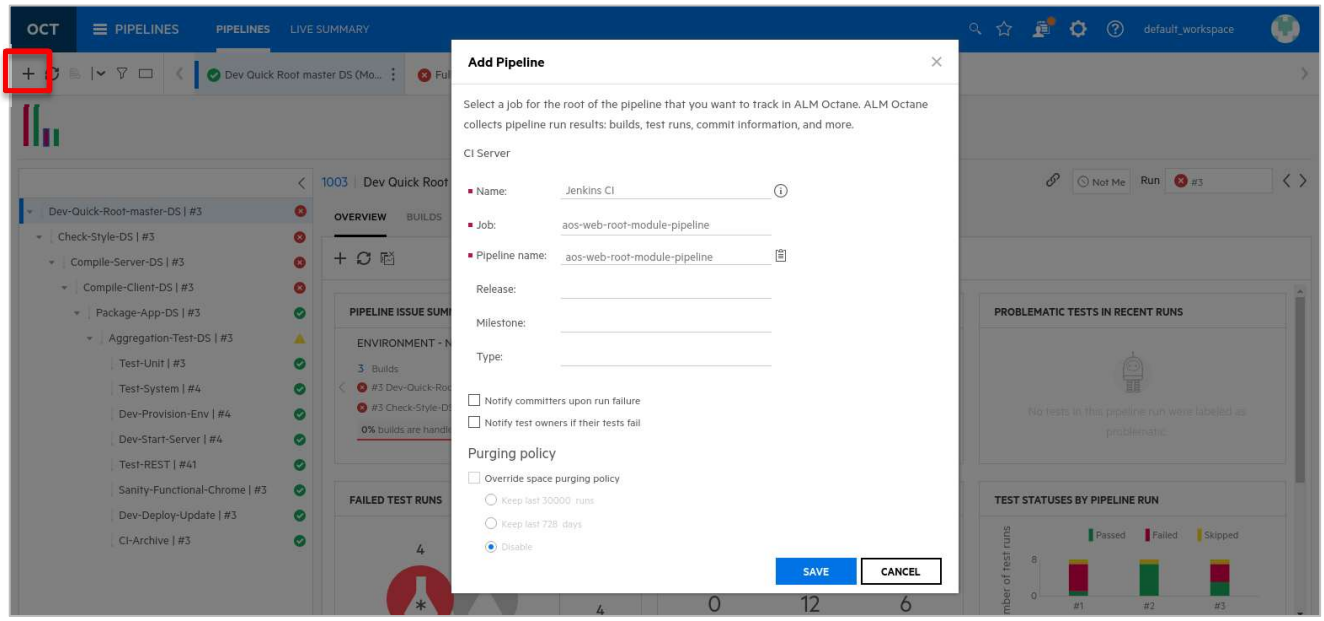
In this section we'll connect a Jenkins pipeline to Octane and then run our pipeline directly from Octane.

1. Open a browser to the **Octane** shortcut and login.
2. Select the **Pipelines** module from the main tool menu.

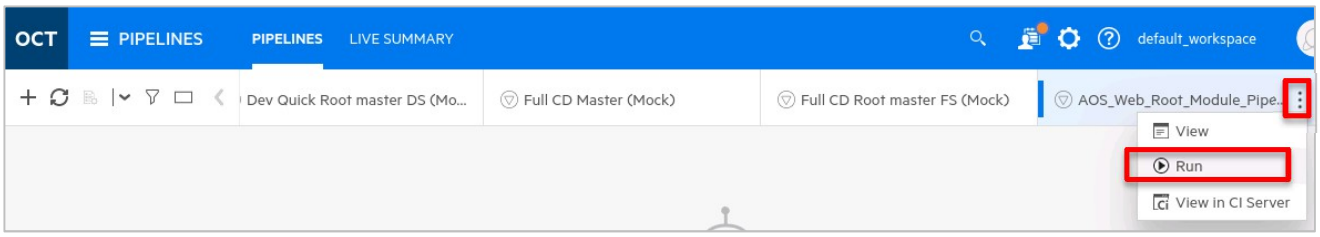


3. Select the plus sign (+) on the left and add a new pipeline with the values below:

   Name: **Jenkins CI** (or the default value)
   Job: **aos-web-root-module-pipeline**
   Pipeline Name: **aos-web-root-module-pipeline**

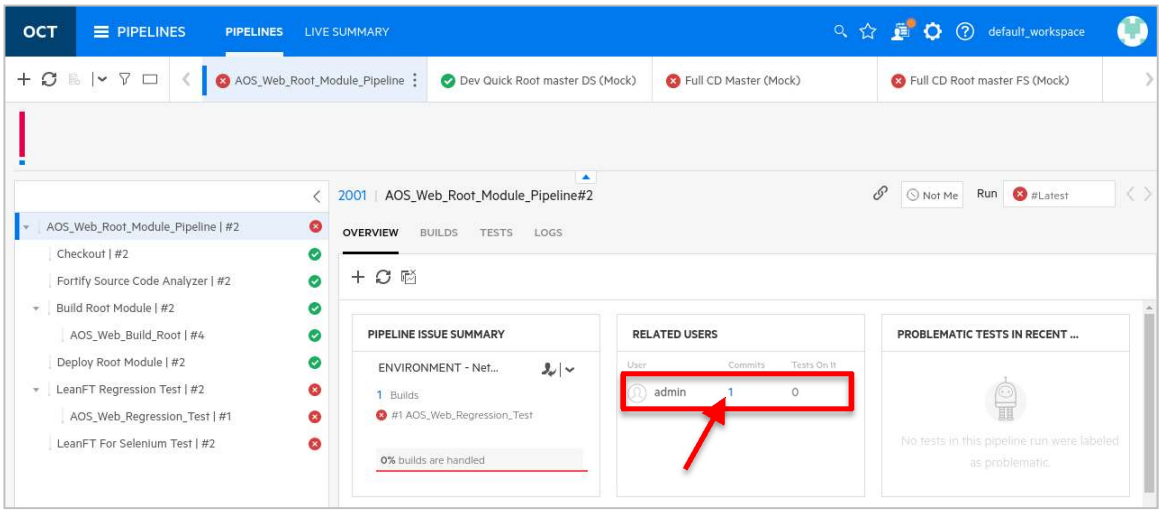


4. Click **SAVE**.
5. In the resulting window, select the new pipeline and click the **Menu Action** button and choose **Run** to start running the pipeline. The pipeline will start to execute in Jenkins.



6. Switch to the **Jenkins** tab and verify that the job is started and progressing.

7. Switch back to Octane and wait for the job to finish and then examine the results in the Pipelines tab. You may need to refresh your view and wait a few minutes to get the results viewable in Octane.

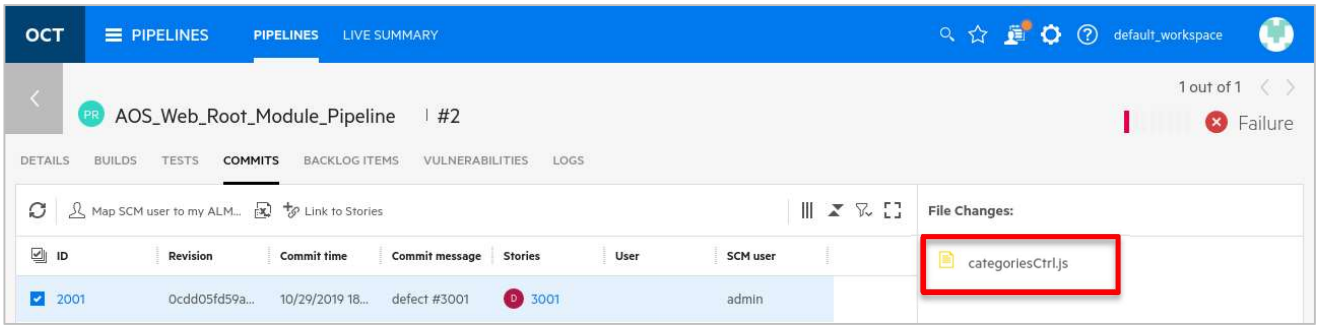


Viewing Code Changes in Octane

1. This build will fail during the regression tests. This is because those tests check that our category was named **Speakers** and now it's called **Audio**. Eventually we'll need to fix the regression test too.

   In the Overview section, locate the dashboard called RELATED USERS and click on the **1** link.
   This identifies our code commit and was done as admin – we can actually map this to our sa@nga account if we want.

   Select the item listed and note the file that was changed – it should match the file from IntelliJ.



2. Click the option to **Map SCM user to my ALM Octane user**
   This associates your **sa@nga** user with the SCM **admin** user.

Nice!

You have just tracked code commits from IntelliJ to Octane!

Exercise 12: Workin' with Gherkin

ROLE: INTEGRATION OR AUTOMATION ENGINEER

Cucumber is a software tool used by computer programmers for testing other software. It runs automated acceptance tests written in a behavior-driven development (BDD) style. Central to the Cucumber's BDD approach is its plain language parser called Gherkin. ALM Octane stores and manages Gherkin defined tests and integrates with IntelliJ to automate them.

In this exercise we'll look at a LeanFT Gherkin test which is failing and we'll investigate the cause and correct it.

Creating a Gherkin Test in ALM Octane

1. We need to have the following VM's running for this exercise:
   - NimbusServer

2. Start the following Docker containers for this exercise.
   ```
   $ nimbusapp octane:15.0.46.68 start
   $ nimbusapp aos:2.2 start
   $ nimbusapp intellij:2.2.0 start
   $ nimbusapp devops:2.2.1 start
   ```

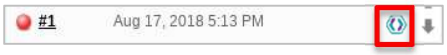
3. If you changed **Speakers** to **Audio**, change it back to **Speakers** and commit and push that change. Then re-run the **AOS_Web_Regression_Test** so it runs successfully.
4. From **NimbusServer**, open a browser tab to the **Jenkins** shortcut.
5. Select the **Test** tab and click on the **AOS_Web_ UFT_Developer_Gherkin_Test** job.

6. Verify that your **UFT Dev Agent** is active from IntelliJ by finding the UFT Dev icon  at the top.

7. Click **Build Now** and watch the console (click the ball icon). Wait for the job to finish. The job should **fail**. This happens because the price in our UFT Dev Gherkins test is **$9.99** and it should be **$15.99.**

Data

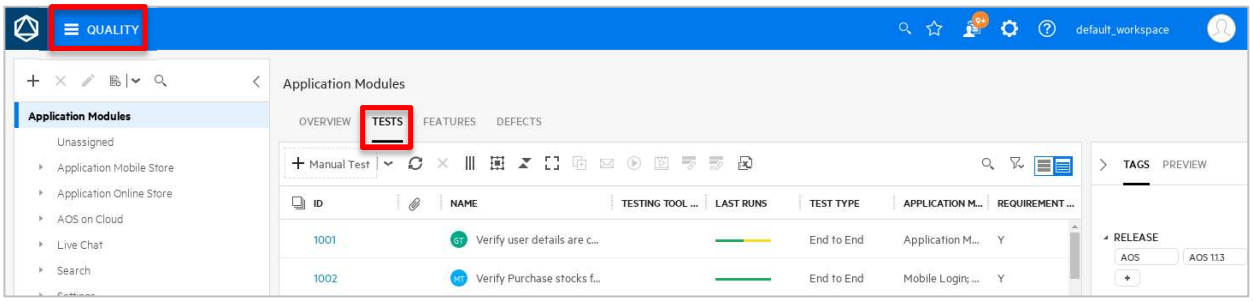
arg0	arg1
$9.99	$15.99

8. To see this result in an HTML report, select the **AOS Web UFT Developer Gherkin Test** and click the **UFT Dev icon** to view the results (right click to open it in a new tab). 

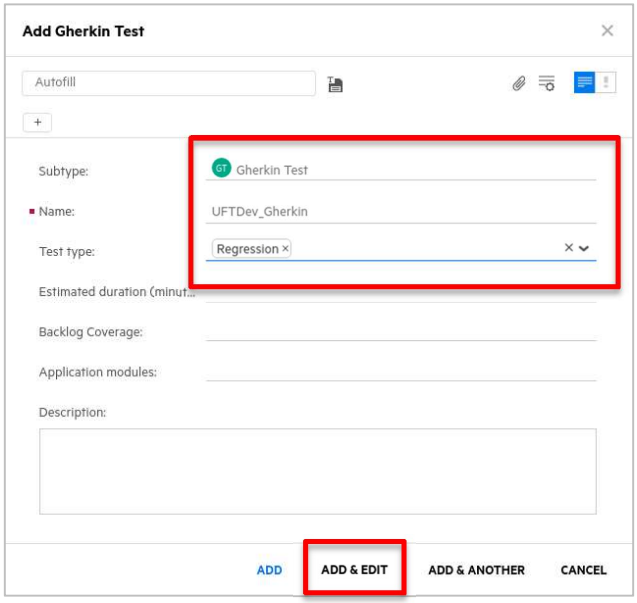
9. The test we ran does not yet exist in Octane. Next, we'll build a new test in Octane and run that. From **NimbusServer**, open a new browser tab to the **Octane** shortcut.

10. **Login** to Octane (sa@nga, Password1).

11. Navigate to the **Quality** module -> **Tests** tab.



12. Select the **+ button** and create a new **Gherkin Test** and name it **UFTDev_Gherkin**.



13. Verify that the **Subtype** is a **Gherkin Test**.

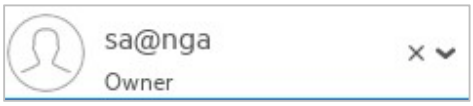
14. Change the **Test Type** to be **Regression**.

15. Click **Add & Edit**.

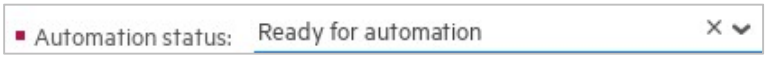
16. **Type** the following Gherkin code in to replace **Feature.**
    Make sure you have **blank lines** between sections.
    Do not change any other section or text.

```
Feature: Advantage Online
   verify multiple scenarios

   Scenario Outline: verify mice color
      Given I am in the site
      And I select the Mice category
      When I filter by "<color>" color
      Then the mouse price is "<price>"

      Examples:
         | color  | price |
         | white  | $29.99 |
         | purple | $15.99 |
```

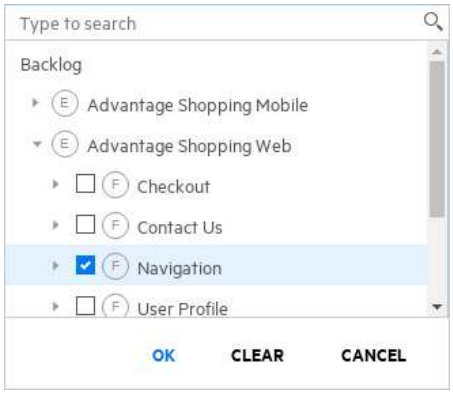
17. **Save** your changes by clicking the floppy disk symbol.
18. Click the **Details** tab and set the **Owner** to be **sa@nga**



19. Change the Automation state to be **Ready for automation**



20. Set the Backlog Coverage to **Advantage Shopping Web -> Navigation** as shown.
    If you can't find this backlog item, select one on your own – it's not critical that you choose this one.

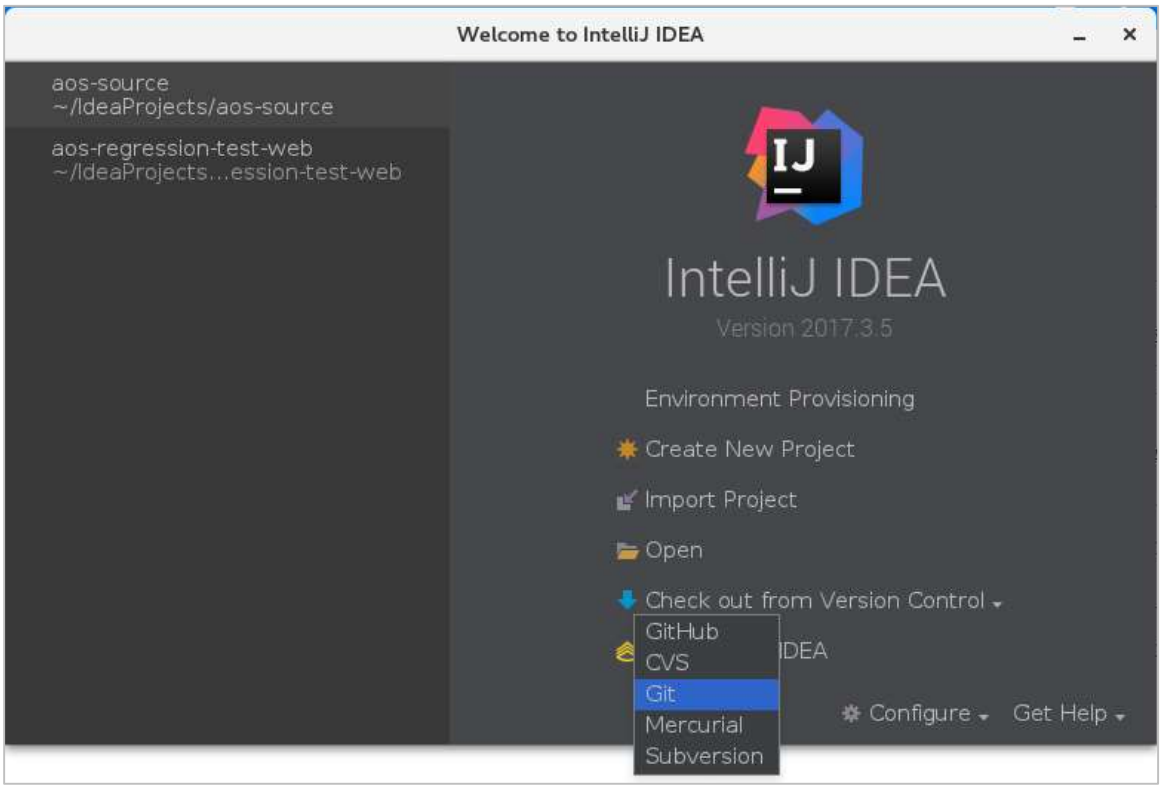


21. **Save** your changes again by clicking the floppy disk symbol.
    By updating these settings, we make this this test visible in IntelliJ as a test needing automation and owned by someone (i.e. sa@nga).

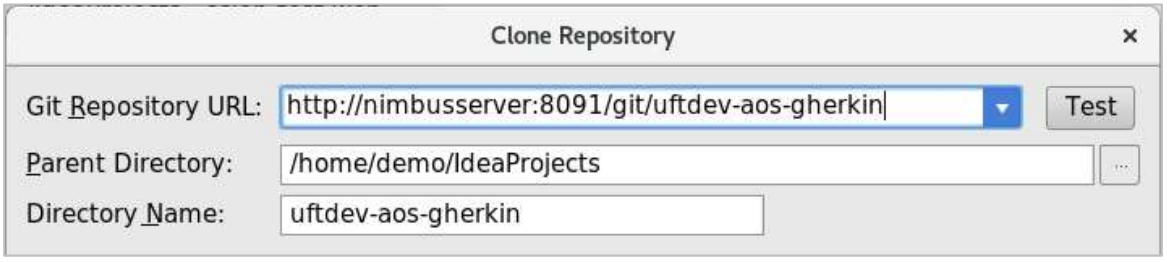
Automating a Gherkin Feature File

Now that someone has defined a Gherkin test for us we need to automate it. Thankfully we won't have the write the code that does this – it's been done for us in an IntelliJ project called **uftdev-aos-gherkin**.

1. Switch to the IntelliJ window.
   Select to **Check Out From Version Control -> Git**

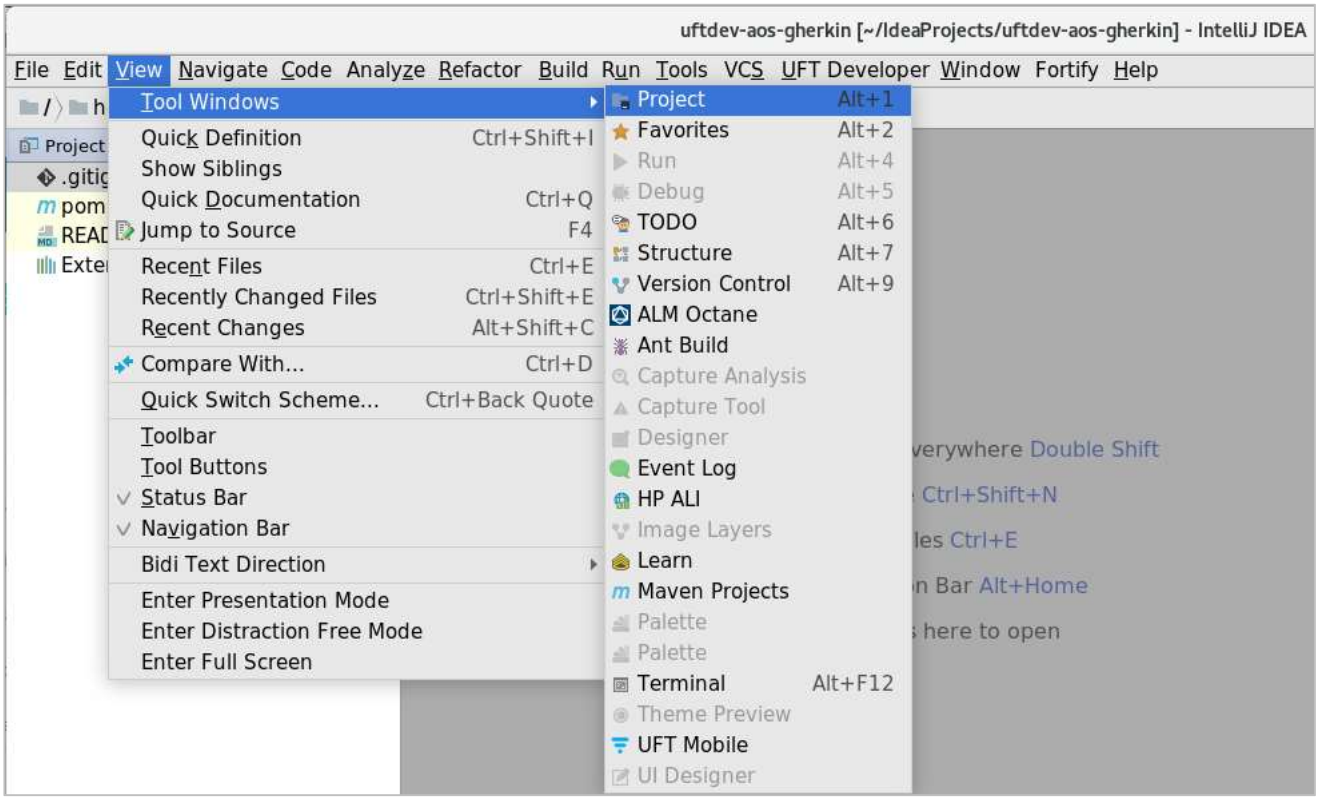


2. Clone the Git repository **uftdev-aos-gherkin.**

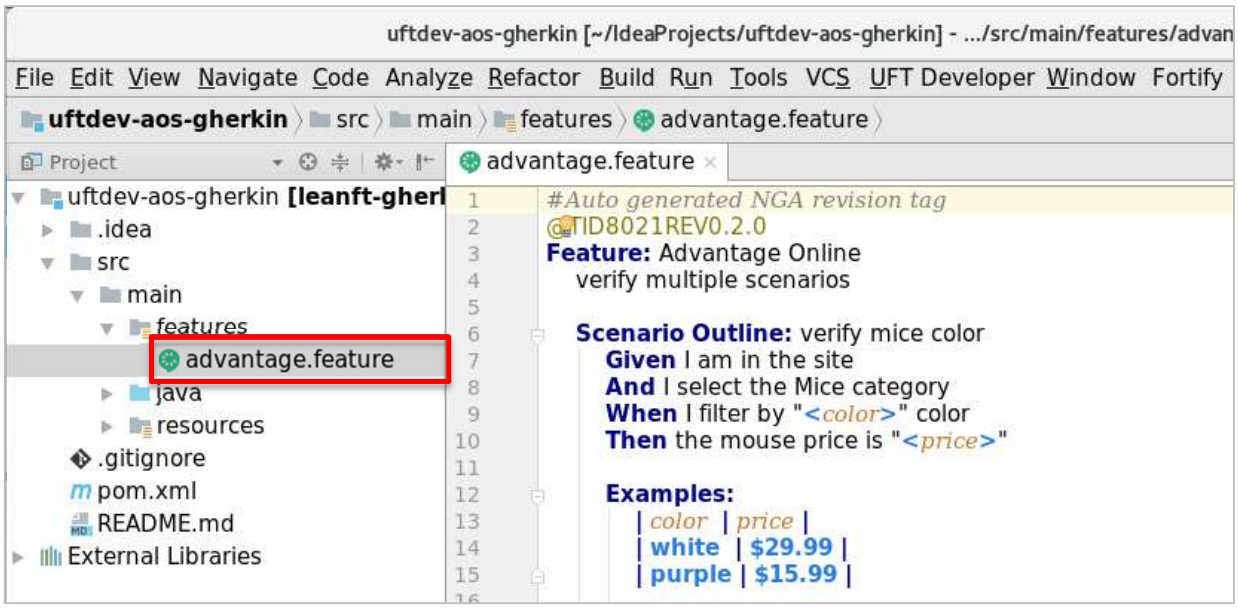


3. Click **Yes** when prompted to **Open** it.

4. Once the project loads, select **View -> Tool Windows -> Project** (or just type **Alt + 1**)
   This opens the project navigation view so you can examine the project structure.

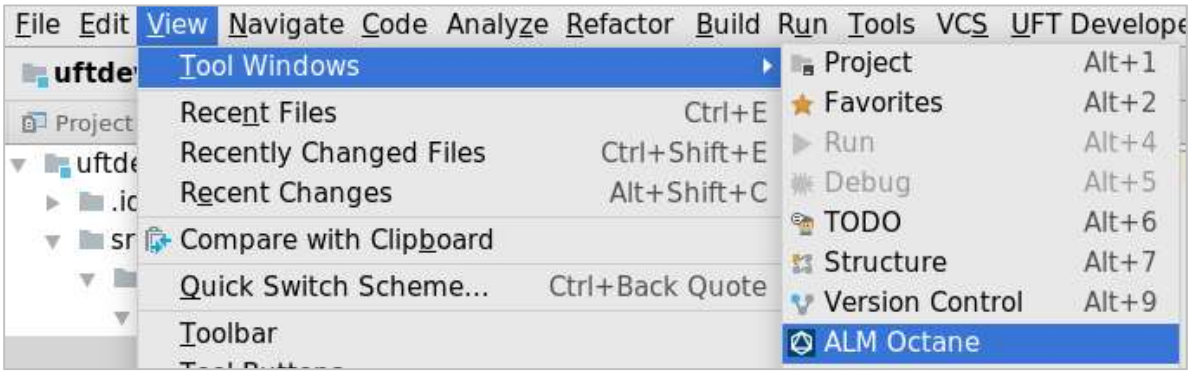


5. In the left side project view, open it up as shown below, exposing **advantage.feature**.
   Double click on **advantage.feature** to see the feature file.
   NOTE – If you don't see the full tree, click the Maven refresh icon.



This **advantage.feature** file is similar to the one we created but is really just a placeholder for us. We'll replace it in the next few steps. For now this just helps us know what we'll eventually be creating. This test failed and if you open AOS and filter for purple mice you'll see that the first price is $15.99.

6. Verify that your Octane window at the bottom is showing and connected to Octane.
   If not, from the IntelliJ top menu select Tool Windows -> ALM Octane.

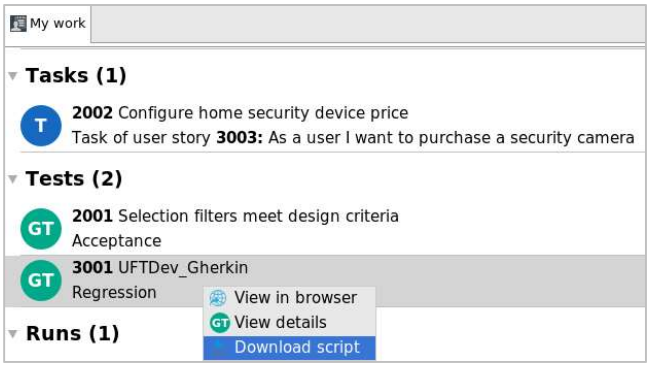


A window at the bottom will open displaying tasks and tests that might need your attention.

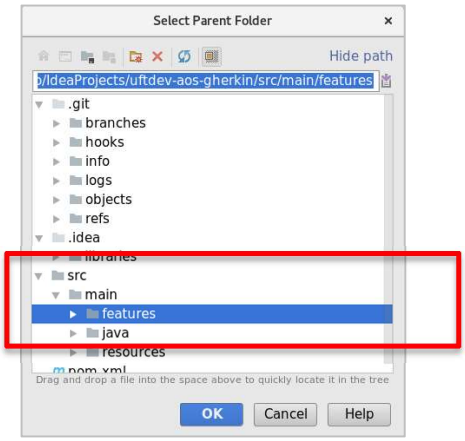
If this windows doesn't connect, **click** the link and enter your Octane credentials and test it.
(http://nimbusserver.aos.com:8085/ui/?p=1001/1002 sa@nga Password1)

NOTE: Sometimes IntelliJ will forget your password when you switch IntelliJ projects.

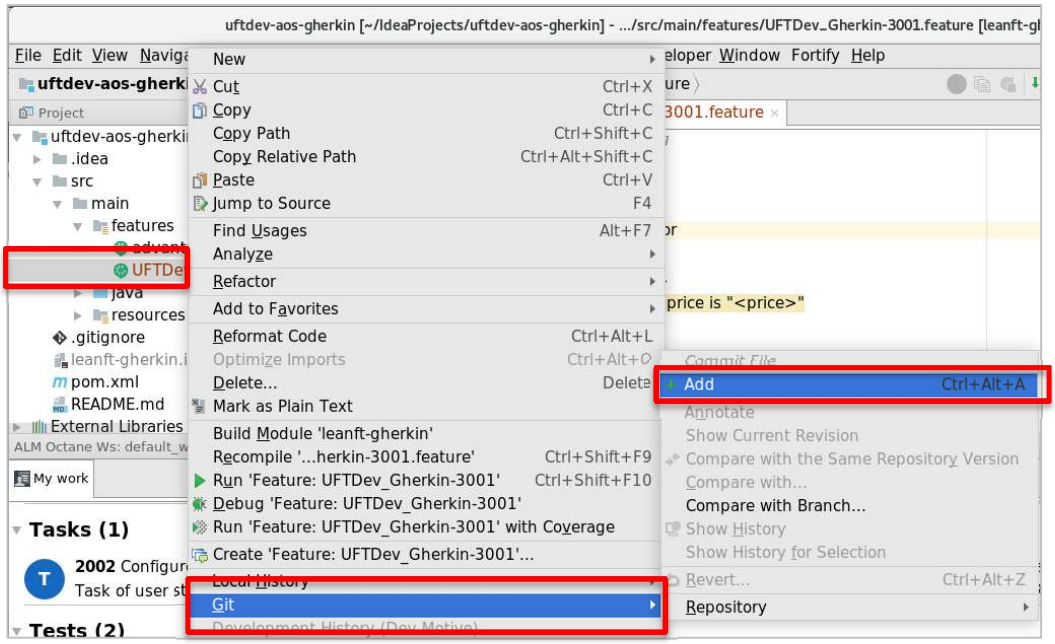
7. In the Octane plug-in window at the bottom, right click on your UFTDev_Gherkin test and select the option **Download Script**.



8. Choose to put the feature file into the **src/main/features** directory – you may need to scroll down. Click **OK**.



9. You'll see a new Feature file appear in the feature folder (in red).
   We now have to put that file under Git control to get it officially part of this project.
   Select the new file and **right click** and choose **Git -> + Add** (the file name will turn green)
   (HINT: You can also do this with CTRL + ALT + A)



10. In the Project tree under **java > net.mf > presales**, double click on the **LeanFtTest** java file.

11. In the code on **line 14**, change the section inside the quotes to read:
    `features="src/main/features"`

    This will allow this and any future feature files to be part of this project.

12. Select the original **advantage.feature** file and **delete** it (we don't need it anymore).

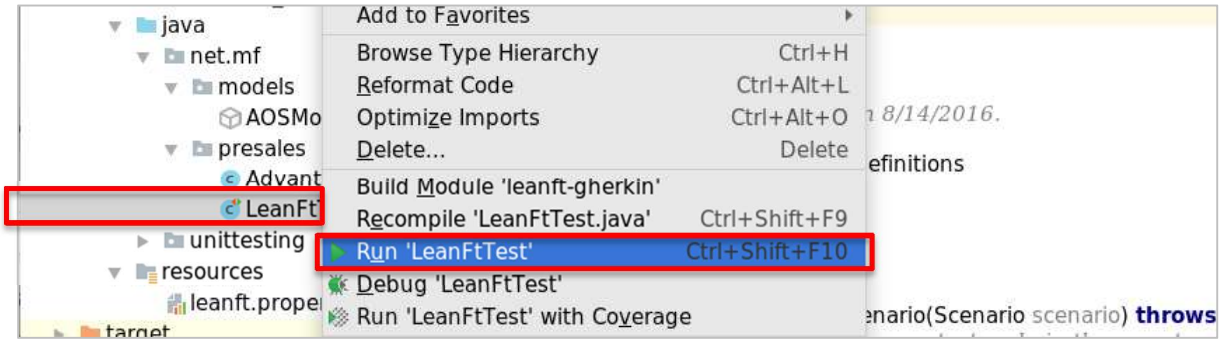
> **NOTE:** The ID that's attached to the feature file name allows it to be tracked to Octane.
> You can stay with this **new** feature file for all your future demos.

13. Select **VCS -> Commit** and enter "**Updated Octane feature file.**" as the commit message and hover over the **Commit** button and choose **Commit and Push**. Set admin@default.com as the email address and ignore any warning messages.

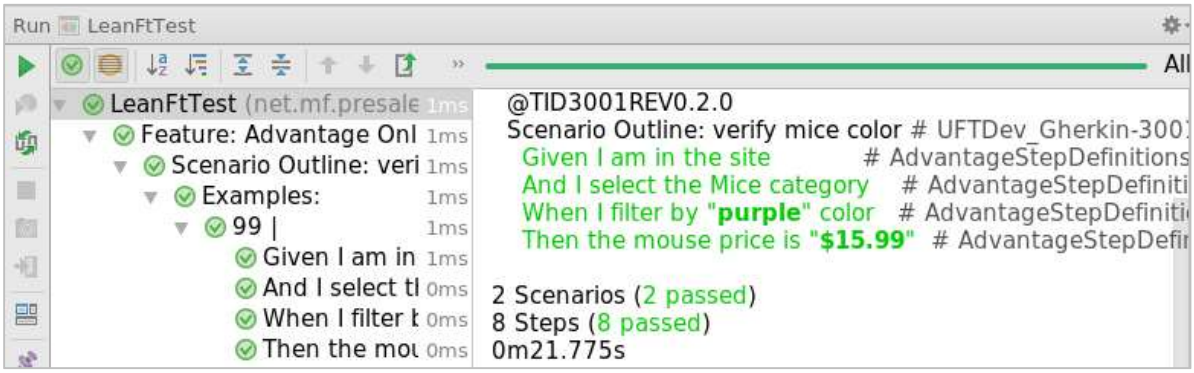
14. In the **Push Commit** window select **Push**. Ignore any items about Warnings.

15. You should see a **Push Successful** message.

16. To verify that your changes worked, **right click** on the **LeanFtTest** under java>net.mf>presales and select **'Run LeanFtTest'** (or select Ctrl + Shift + F10)

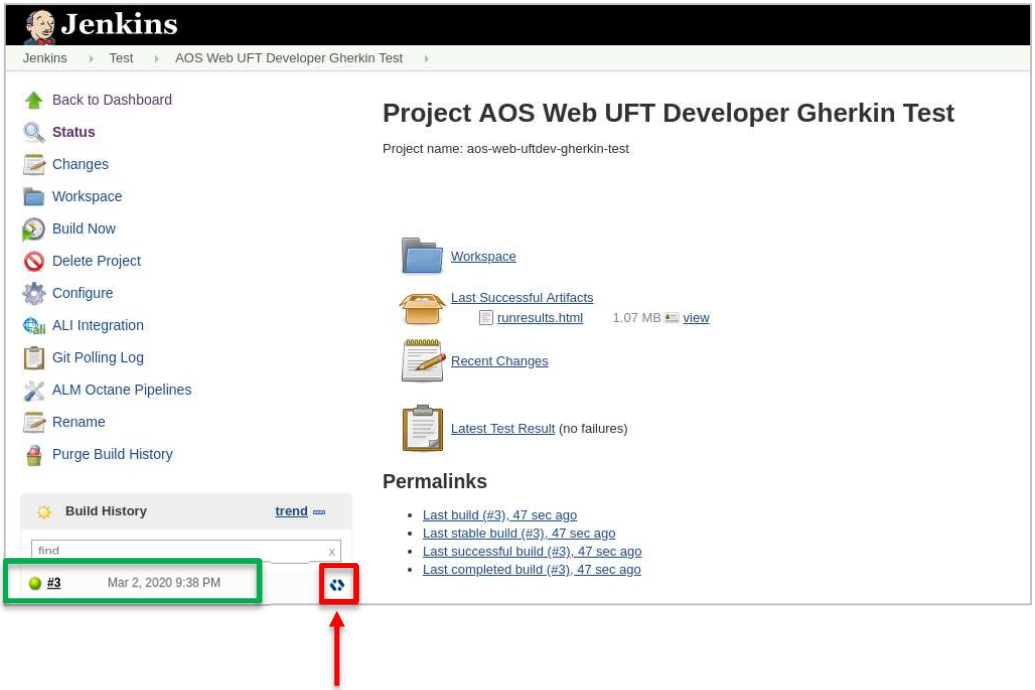


17. An execution window will open at the bottom and then a browser will open and start running the test. If everything went fine it should run two scenarios and they should both pass.



18. Switch back to Jenkins and run (build) the **AOS_Web_UFT_Developer_Gherkin_Test** job.
    It should run **successfully** this time!

> **What did we just do?**
>
> By swapping the feature file and linking it to Octane, we essentially replaced the original Gherkin test (that wasn't stored in Octane) with one now managed by Octane. Going forward, we can make changes in Octane to the Gherkin code and it will sync them with the test that runs in Jenkins.



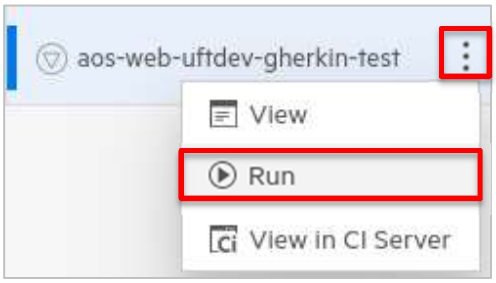
19. Click on the **UFT Dev RunResults** icon to see the detailed UFT Dev report for your Gherkin test.

20. To get our results into Octane we need to create a pipeline for it.
    Switch back to **Octane** and select the **Pipelines** module.

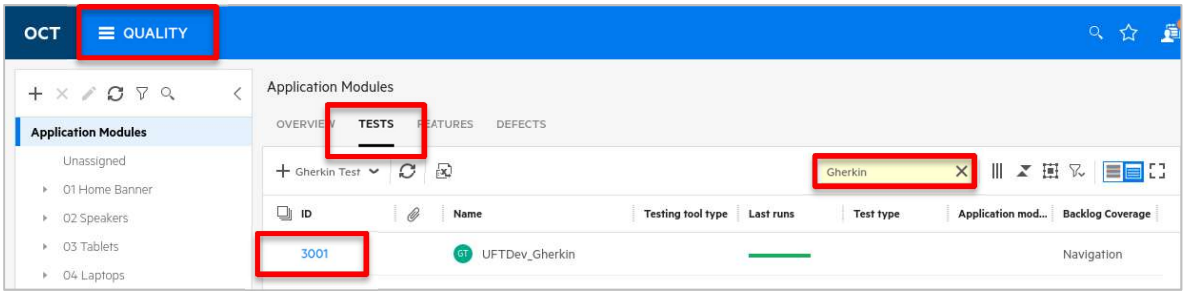
21. Click the **+** sign and add the **AOS_Web_UFT_Dev_Gherkin_Test** pipeline.
    Name: **Jenkins CI**
    Job: **aos-web-uftdev-gherkin-test**
    Pipeline name: **aos-web-uftdev-gherkin-test**

    Click **Save.**

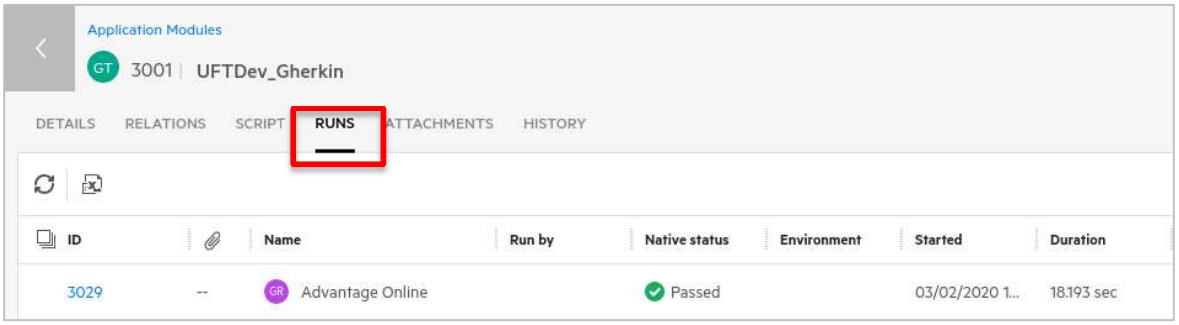
22. **Select** and **Run** the pipeline (it should pass).



23. **Navigate** to the **Quality -> Tests** module in Octane and locate your test (probably at the bottom). Try searching for **Gherkin**



24. **Locate** your test and click on its numeric link to see your results (in the **Runs** tab).



Exercise 13: Creating Testing Jobs in Jenkins

ROLE: INTEGRATION OR AUTOMATION ENGINEER

Projects (or jobs or builds) and pipelines form the basis of what Jenkins manages. In this section we'll automate running a UFT One test from both a file location and from within ALM QC. Normally you would pull a file-based test from an SCM but for this example we've not included that.

NOTE: This exercise requires a UFT One script to be located in
C:\Users\demo\Documents\Unifie~1\AOS_UFT_WEB_Regression and that the
UFT Dev Agent is NOT running on NimbusClient (since we're using UFT, only one test type at a time).

Running a UFT One test from a file

1. We need to have the following VM's running for this exercise:
   - NimbusServer
   - NimbusClient

2. Start the following Docker containers for this exercise.

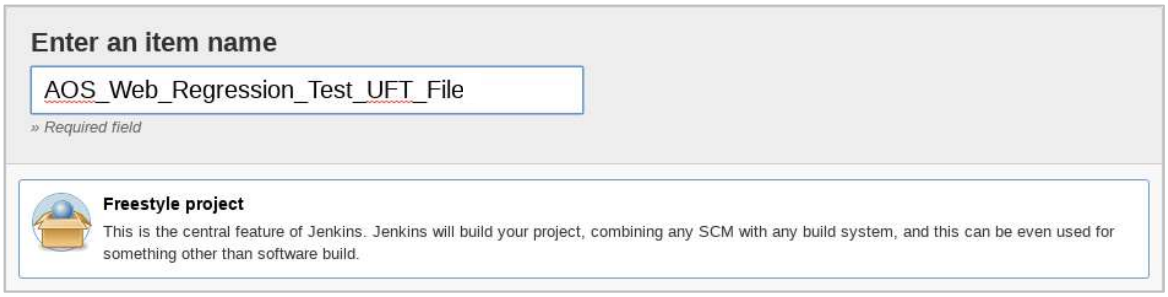
```
$ nimbusapp devops:2.2.1 start
$ nimbusapp aos:2.2 start
```

3. From a browser on **NimbusServer**, open the Jenkins shortcut.
   Verify that your connection to **nimbusclient_uft** is enabled and looks like this:

   The four **idle** connections indicate that you can run up to four Jenkins jobs at once.

4. From the **Jenkins** main dashboard, select **New Item**.

5. Select **Freestyle project** and enter the item name: **AOS_Web_Regression_Test_UFT_File**



6. Click **OK**. This will save the file and bring you into the **Configure** section of this job definition.

7. In the **General** section select the checkbox **Restrict where this project can be run** and type **nimbusclient_uft** (lowercase) into the label expression.
   You may need to backspace the last space character for Jenkins to accept the text.

   By restricting where this job is run, we are ensuring that this UFT test runs on the Windows platform (NimbusClient) since UFT One tests must run on a Windows machine.

8. Scroll down to the **Build** section and select **Add Build Step**
9. Select **Execute Micro Focus tests from file system**

10. In the **Tests** box type:
    C:\Users\demo\Documents\Unifie~1\AOS_UFT_WEB_Regression

11. In the **Post-build Actions** section, click **Add post-build action** and select **Publish Micro Focus tests result** and set the value to be **Always Archive test reports**.

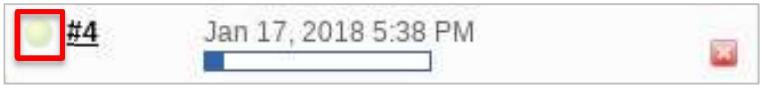
12. Click **Save** to save your changes.

> **What did we just do?**
>
> We've just created a simple Jenkins job that will run a UFT test on NimbusClient and save the HTML results so that it's viewable in Jenkins. This job could now be part of a larger pipeline of activities.

13. From Jenkins, click **Build Now** on the left side.

This will start running your job which executes a UFT test on the NimbusClient VM. If you like, you can switch to the **NimbusClient VM** and watch the test execute.

14. As the test runs you'll see a flashing ball symbol and a progress bar.



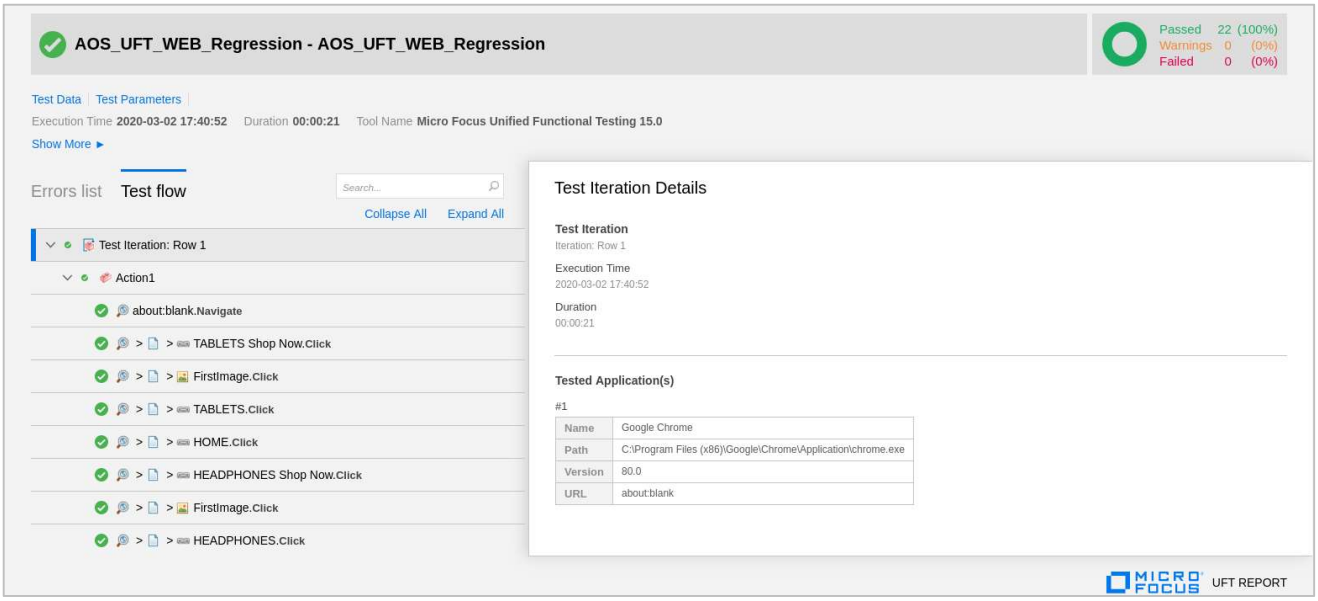
Click on the flashing ball to see a textual console description of what's happening.

```
----------------------------------------------------------------------------------------------------
================================================
Run status: Job succeeded, total tests: 1, succeeded: 1, failures: 0, errors: 0
Passed : C:\AOS_UFT_Regression
================================================
Recording test results
Report archiving mode is set to: ALWAYS_ARCHIVE_TEST_REPORT
add html report info to ReportInfoToCollect: [date]17/01/2018 10:38:34
begin to collectAndPrepareHtmlReports
collectAndPrepareHtmlReports, collecting:C:\AOS_UFT_Regression\Report
workspace: C:\Jenkins\workspace\AOS_Web_Regression_Test_UFT_File
source: C:\AOS_UFT_Regression\Report
copy from slave to master:
/var/lib/jenkins/jobs/AOS_Web_Regression_Test_UFT_File/builds/4/archive/UFTReport/UFT_Report_HTML_tmp.zip
UnzippedFolderPath is:
/var/lib/jenkins/jobs/AOS_Web_Regression_Test_UFT_File/builds/4/archive/UFTReport/Report targetPath is:
/var/lib/jenkins/jobs/AOS_Web_Regression_Test_UFT_File/builds/4/archive/UFTReport/AOS_UFT_Regression
set the report urlName to artifact/UFTReport/AOS_UFT_Regression/run_results.html
Adding a report action to the current build.
Finished: SUCCESS
```

15. After the console data finishes, click on the **Last Successful Artifacts**

16. Next, click on the left side **UFT Report** to see the **HTML Run Results** report.

17. Right Click on the **AOS_UFT_Regression** link to have Jenkins open the report in a new tab.



Great!

You've run a UFT One Test from Jenkins on a different machine!

Running a UFT One test from ALM QC

In this section we'll configure Jenkins to run a test that's stored in an ALM QC project.
Initially we'll copy the test we just used into the AOS project tree and then we'll set up a Jenkins job to run it.

Setting up ALM

1. Start the ALM container by running the following command from **NimbusServer**:

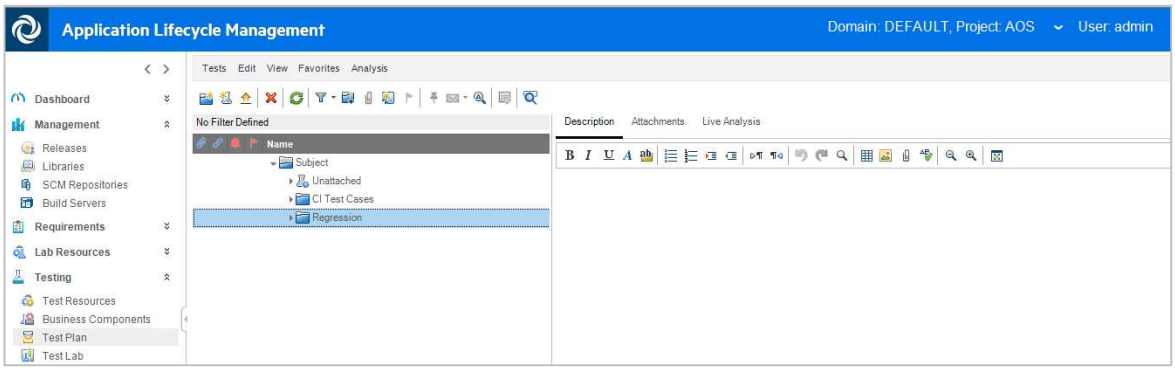
   ```
   $ nimbusapp alm:15.0.1 start
   ```

2. **Wait** for ALM to finish booting up (watch your **System Monitor**).

3. From **NimbusClient**, open **Internet Explorer** and select the **ALM/QC** shortcut from the **ADM Products** folder shortcut.

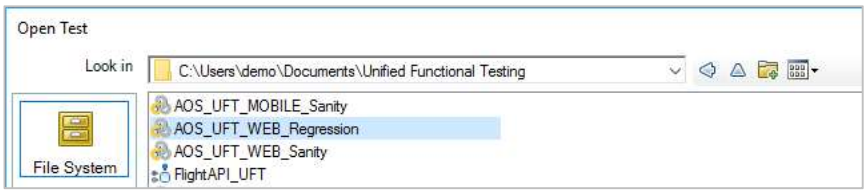
4. Select the **ALM Desktop Client** link and enter your ALM QC login credentials (admin, Password1) and select to connect to the **DEFAULT** domain and **AOS** project. (NOTE: Not the **AOB** project)

5. From **IE** in the left side menu select **Testing -> Test Plan** and create a new **Folder** under the **Subject** folder called **Regression**.



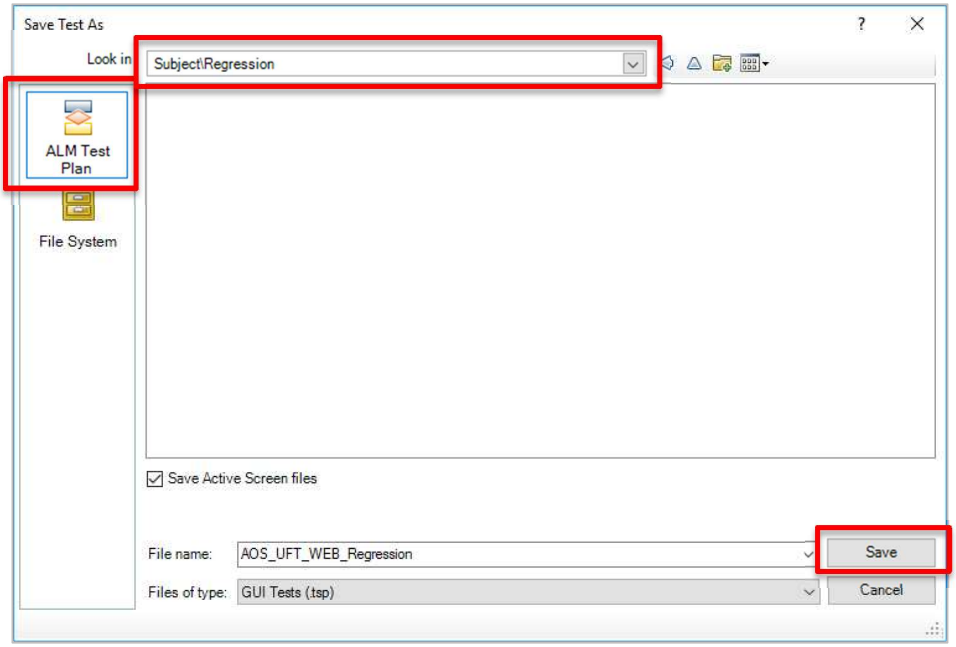
6. Start **UFT One** from the desktop shortcut (**ADM Products** folder > **Micro Focus Unified Functional Testing**) and select just the add-in for **Web** and open the following test:

   C:\Users\demo\Documents\Unified Functional Testing\AOS_UFT_WEB_Regression.

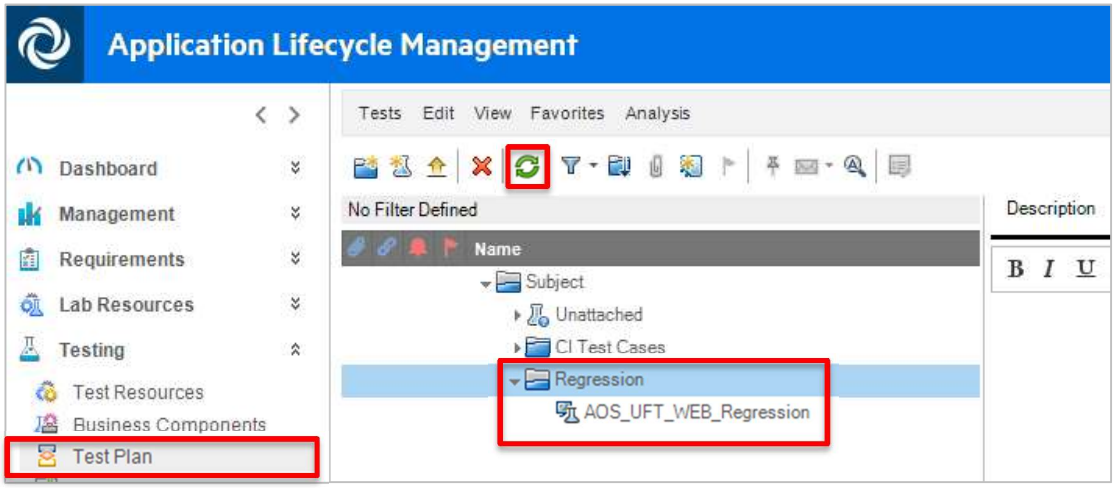


7. From **UFT One**, select **ALM -> ALM Connection** from the menu and enter the same credentials (admin, Password1, DEFAULT, AOS) and then clock **Close**.

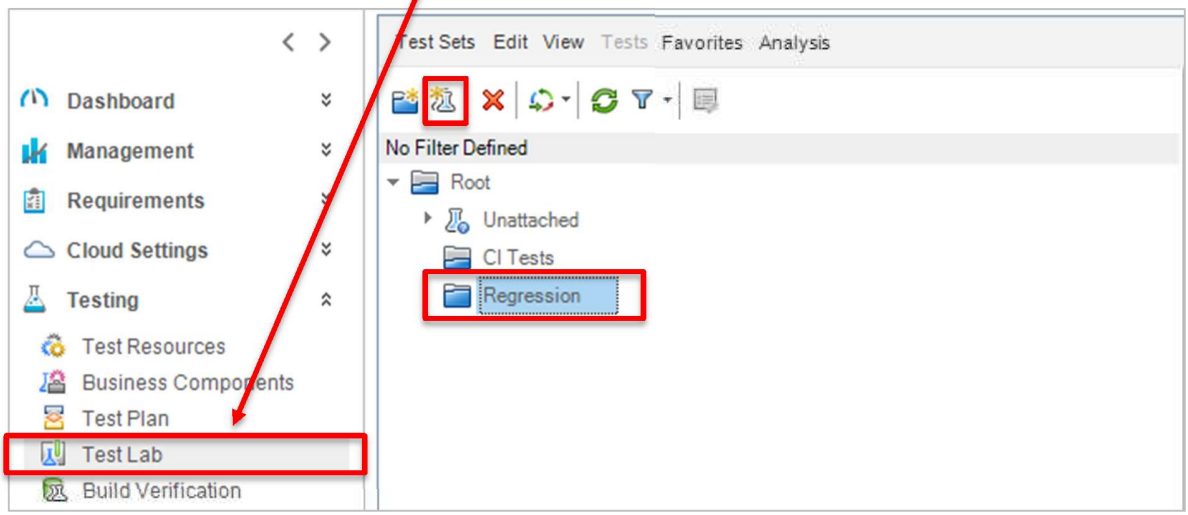
8. From **UFT One**, select **File -> Save AOS_UFT_Web_Regression as** into ALM in the **Regression** folder in the test plan (see screenshots below).



If you switch back to ALM it and refresh it, it should look like this:

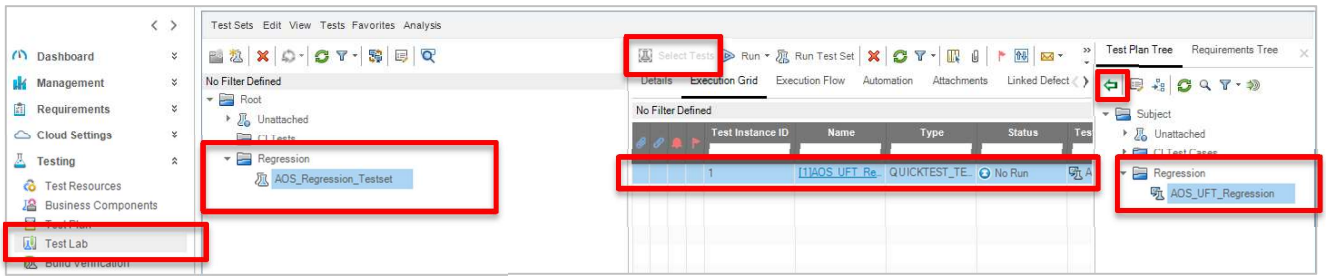


9. From ALM QC in IE select the **Test Lab** module and create a folder under the **Root** folder called **Regression**. This is different from what you did before.



10. Under the **Regression** folder, create a **Test Set** called **AOS_Regression_Testset**.

11. Click **Select Tests** and **add** the **AOS_UFT_Regression** test to the **AOS_Regression_Testset** as shown.



Setting up Jenkins

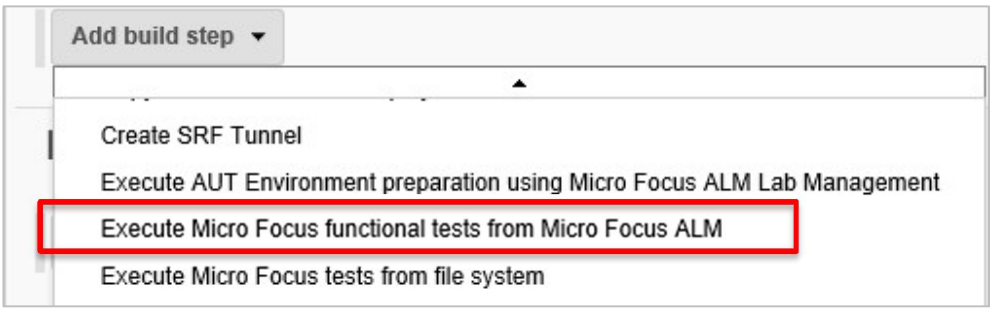
1. At this point you can **close UFT One** and **Internet Explorer**.
2. Open a **new tab** in **Chrome** on **NimbusServer** and select the **Jenkins** shortcut.
3. From the Jenkins main dashboard, select **New Item**.

4. Select **Freestyle project** and enter the item name: **AOS_Web_Regression_Test_UFT_ALM**

5. Click **OK**. This will save the file and bring you into the **Configure** section of this job definition.

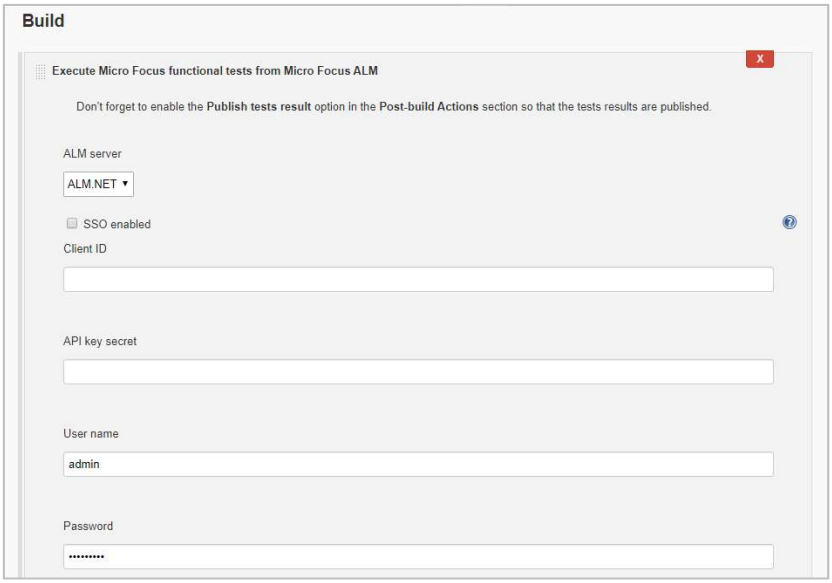
6. In the General section select the checkbox **Restrict where this project can be run** and type **nimbusclient_uft** (lowercase) into the label expression.

7. Scroll down to the **Build** section and select **Add Build Step.**
Select **Execute Micro Focus functional tests from Micro Focus ALM**



8. **Set** the following fields:

Client ID: **<leave blank – despite the warning>**
Client Secret: **<leave blank – despite the warning>**
User name: **admin**
Password: **Password1**
Domain: **Default**
Project: **AOS**
Test Sets: **Root\Regression\AOS_Regression_Testset**



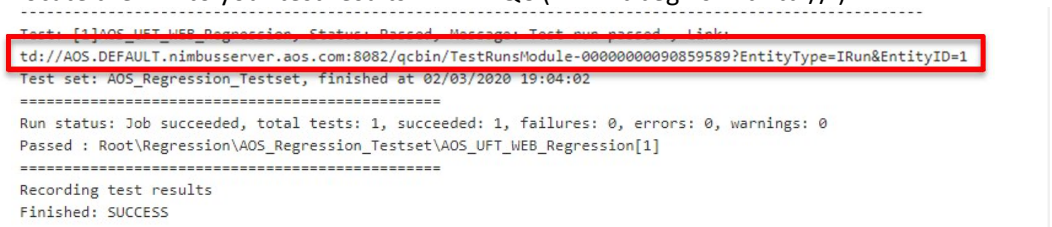
9. In the Post-build Actions, add the **Publish Micro Focus tests result** and set the **Report Archive** mode to **Always archive test reports**.

10. From the Jenkins dashboard, select the **Test** view tab and locate your new job.

11. Click on the **AOS_Web_Regression_Test_UFT_ALM** link and select **Build Now**.
This will start running your job which executes the UFT test from ALM on the NimbusClient VM.
If you like, you can switch to the **NimbusClient VM** and watch the test execute.

12. As the test runs you'll see a flashing ball symbol and a progress bar.

13. **Click** on the flashing ball to see a textual console description of what's happening.
Locate the link to your test results in ALM QC (HINT it begins with td://)



Copy and paste that link into Internet Explorer to see your results!

Bonus Question #1

How can you find the results of a previous build?

Bonus Question #2

Where can you see the HTML results of the test run from ALM?

Challenge Exercise #1

Create one Jenkins project that runs both the UFT One from File and UFT One from ALM tests.

Exercise 14: Integrating UFT One tests in Octane

ROLE: QA MANAGER, INTEGRATION OR AUTOMATION ENGINEER

Octane can integrate with existing UFT tests that are stored in Git (either locally or in GitHub) and be able to run those tests and track their results. This exercise explains how to configure this integration and execute UFT tests from Octane as a pipeline build. *This exercise was originally developed on Octane 12.55.25.114.*

Configuring Jenkins for UFT execution

Jenkins acts as the script executor when running UFT tests from Octane. In order for Jenkins to run UFT tests it needs to connect to a windows machine with UFT One installed on it. Jenkins also need an execution node configured for that machine. These have already been configured for our Nimbus environment.

1. We need the following VM's to run this exercise:
   - NimbusServer
   - NimbusClient

2. Start or verify the following running Docker containers for this exercise.

```
$ nimbusapp octane:15.0.46.68 start
$ nimbusapp aos:2.2 start
$ nimbusapp devops:2.2.1 start
```

3. From the **NimbusClient**, open a browser and open the **Jenkins** shortcut link.

4. Verify that the **nimbusclient_uft** node is online.

If it shows 

, **reboot** the NimbusClient machine and check it again (it has a 60 second delay after booting). If you stil have issues getting nimbusclient_uft to be online, check with your instructor.

Configuring a Git repository for UFT tests

In order for Octane to manage UFT tests they must be stored in either a local or remote Git repository. Luckily, our devops image has just such a repository already located at http://nimbusserver:8091/git

In this section we'll create a new, empty repository (bare) in the devops container and then clone that to NimbusClient. This creates a remote repository where we can store UFT tests. Then we'll copy some pre-existing tests into that directory and push them into the Git repository on devops.

Git is actually a fairly simple, yet powerful version control system. Once you initialize a Git repository (it's just a directory), anything you place in that directory can and will be managed by Git. You just have to run a commit command to place them under Git's control.

There are two types of Git repositories – local and remote. Typically, you "pull" from a remote repository and you get a local repository where you can work and make changes. When you're ready to share those changes, you "push" them back to the remote so everyone else can see them.

Basic Git commands:

Command	Description
`git init`	Creates a git repository in the current directory. All files and directories (recursively) in that directory can be managed by git. Git places a .git folder in this directory and stores its information in there.
`git init --bare`	Creates an empty git repository. Usually used to create an empty remote repository which is cloned elsewhere and then files added locally and pushed to the remote.
`git add myfile`	Git adds the file "myfile" to its repository of tracked files. This file is called "staged" since it's ready to be committed.
`git commit –m "message"`	Git commits all the "staged" files and create a "branch". Once a commit is done, the directory and Git's snapshot of the directory are in sync. Git requires a message (describing what was done) to be supplied.
`git remote add origin URL`	Git assigns a remote repository URL to the local repository for pushing and pulling.
`git push`	Git pushes its latest snapshot of the repository to a remote repository.
`git pull`	Git pulls files and directories to your local repository
`git clone`	Git creates a copy of a remote repository and remembers and sets the remote repository location.
`git status`	Git shows the status of the current repository and lists the files in staging.

1. On the NimbusServer VM, open a terminal window.

2. From the terminal window type the following commands (do not type the prompt commands - $ or #):

```
$ docker exec –it devops bash        (or $ de devops)
# cd gitrepo
# git init --bare UFT_Tests.git
# chmod -R 777 UFT_Tests.git
```

   NOTE: The UFT-Octane integration insists that the remote directory ends in .git.
   That's not all bad since naming a Git repository ending in .git is a recognized common practice.

3. Switch to the **NimbusClient** VM and open up a Command Prompt (cmd) window.

4. Change directory to the C:\ directory with following command:

```
C:\Users\demo> cd c:\
```

5. Type the following command:

```
C:\> git clone http://nimbusserver:8091/git/UFT_Tests.git
```

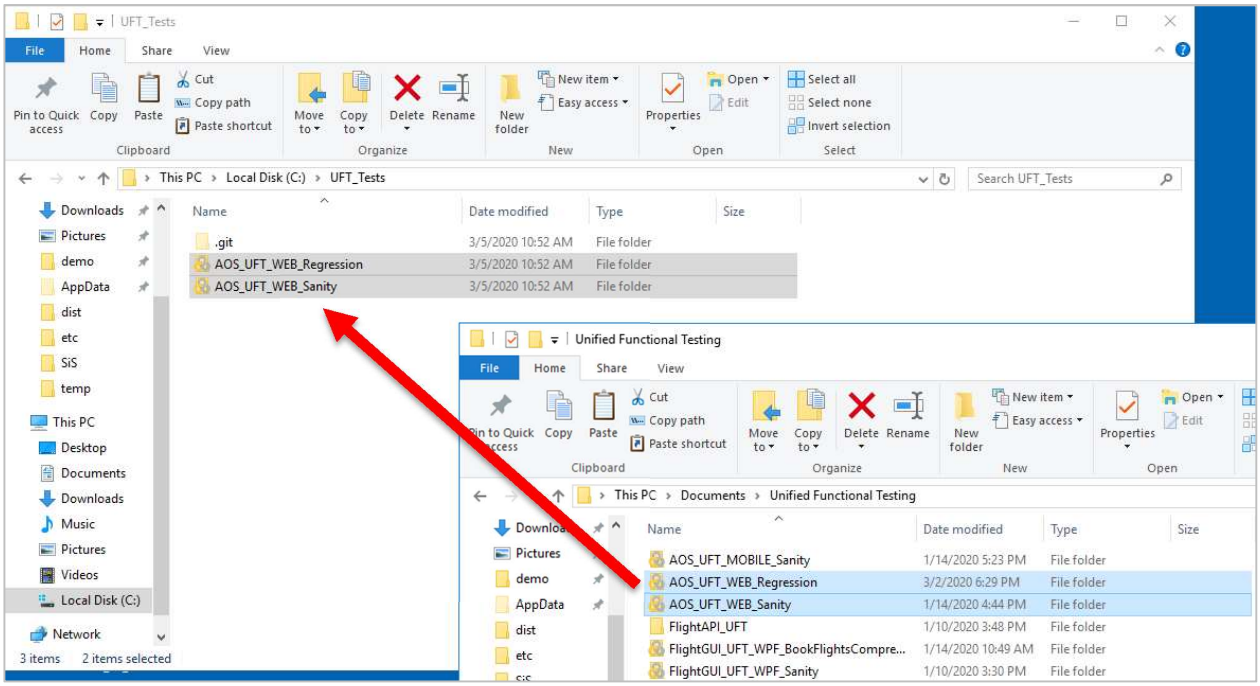
   This command does two things:
   1. Creates a copy of the empty directory in this remote repository
   2. Sets the remote locations for this local repository to be the specified URL.

6. Open an Explorer window and navigate to **C:\Users\demo\Documents\Unified Functional Testing**

7. Select the following two UFT projects and select COPY:
   - AOS_UFT_Web_Regression
   - AOS_UFT_Web_Sanity

8. Navigate to the **C:\UFT_Tests** folder that was just created in step 5.
   NOTE: When Git clones this repository it removes the .git extension

9. Paste the two projects into this folder as shown:



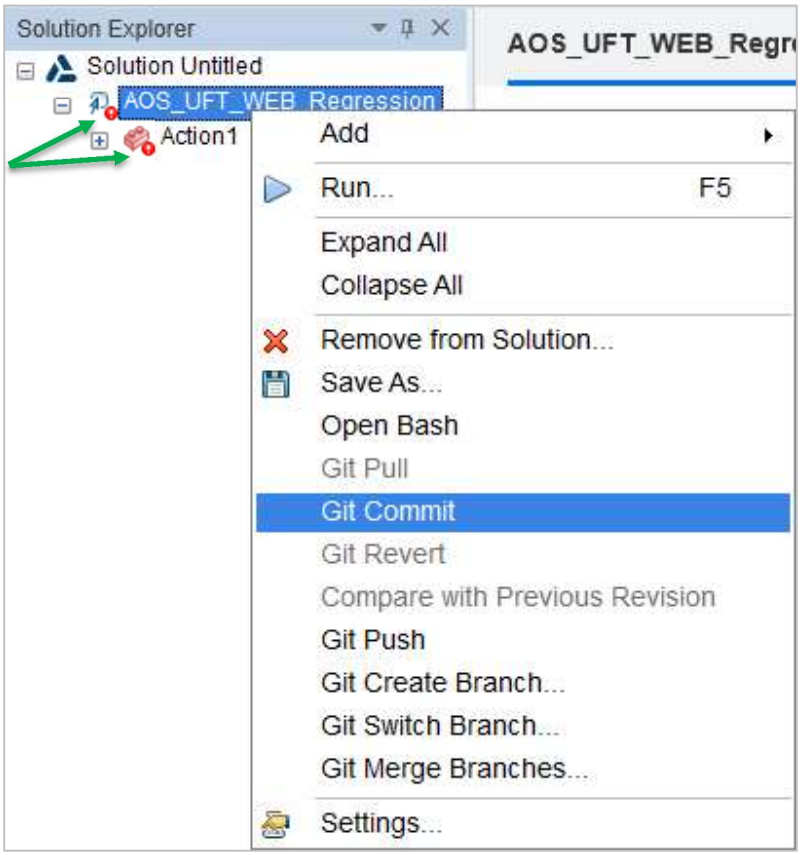
10. Open UFT One and load the **Web** add-in.

11. Open the UFT One project: **C:\UFT_Tests\AOS_UFT_Web_Regression**

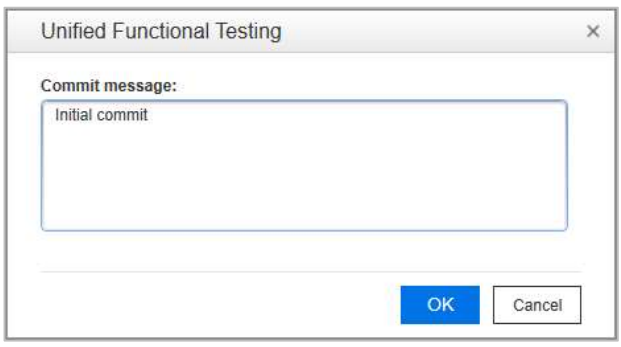
12. From the **View** menu select the **Output** option. At the bottom of your UFT One window you'll now see the **Ouput** tab. This window will allow you to view the Git commands as they are executed.

13. Notice below the small **red** symbols on this project that indicate that this project is not yet in sync with Git. These will turn green when the project has been committed.

    From the **Solution Explorer** panel on the left, select the **AOS_UFT_Web_Regression** project at the top and right click to bring up the project options and select **Git Commit**.



14. In the Commit message text box type “Initial commit” and click **OK**.
    You should see status messages in the Output window and your project icons should turn green.



15. Again, select the **AOS_UFT_Web_Regression** project and right click to bring up the project options and this time select **Git Push.** Click **OK** on the warning popup.

    This will push this project to our remote Git repository in the devops container.
    Enter **root** for the Login and **Password1** for the password and click **OK**.

16. In the Commit box, enter “Initial push” and click **OK**

17. Select **File->Close Solution** (or CTRL+Shift+F4) to close this project.

18. Perform steps **13-16** again, substituting **AOS_UFT_Web_Sanity** for the UFT One project name.

19. **Close** UFT.

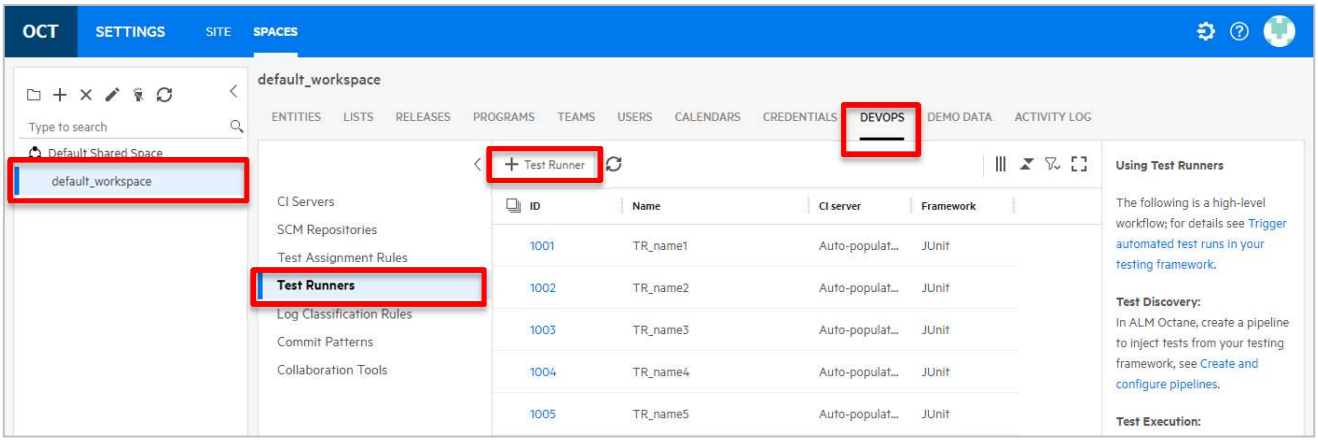
> **What did we just do?**
>
> We created a Git folder on C:\ to hold all of our UFT Tests that are managed by Git. Then we copied two UFT tests into that folder and sync'd (committed) them with Git. Finally, we pushed those two tests to the remote folder we defined when we cloned the Git repository. Next we'll set things up in Octane by creating a creating a Test Suite and adding our tests to that test suite.

Configuring the Octane integration with UFT One

We're almost ready to run our tests in Octane but first we need to tell Octane about the Git repository we created and then we have to create a Test Suite to hold our UFT One tests. We'll do that next.
(Install TDConnect to fix ALM issue)

1. From **NimbusClient**, open a browser tab and select the **Octane** shortcut and login.
   Name: sa@nga
   Password: Password1

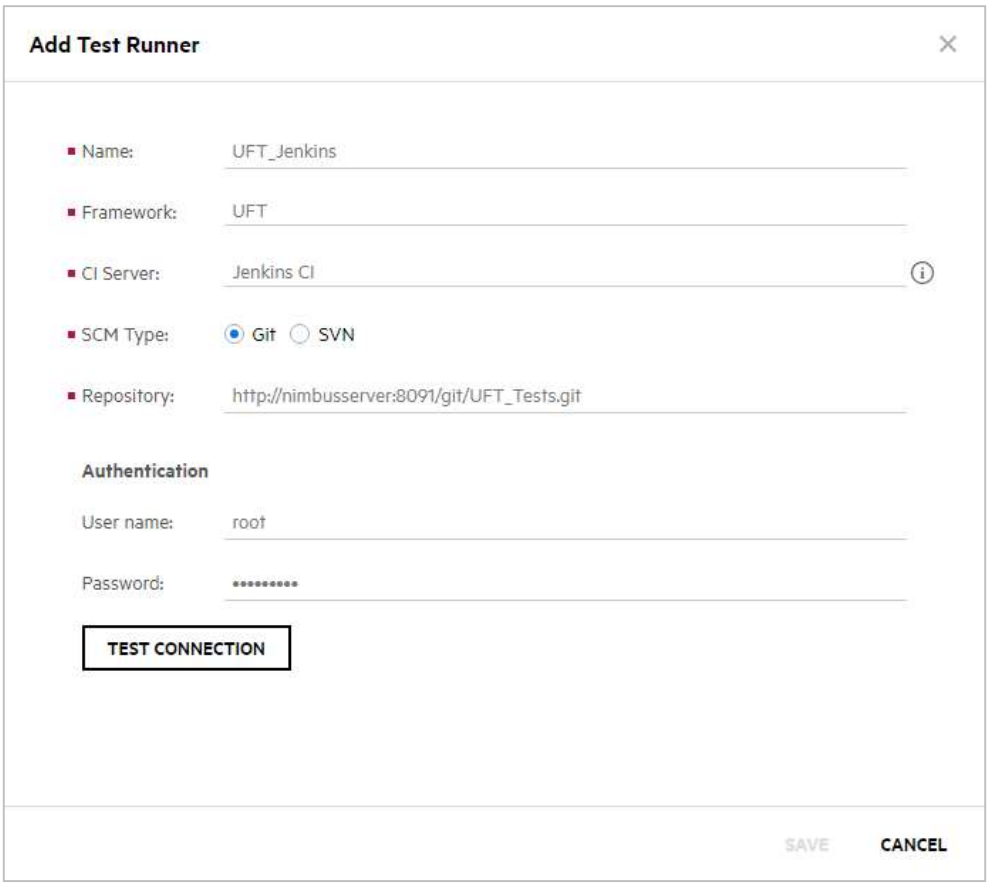
2. Click the **Settings** gear icon  and then click on **Spaces** on the **Administrator** option.
3. Select the **default_workspace** on the left panel.



4. Select the **DEVOPS** tab on the top.
5. Select the **Test Runners** option on the left menu.
   This option allows you to connect to a Git managed UFT One tools repository.
   NOTE: You can only define one testing tool connection per workspace.

6. Click the **+ Test Runner** and fill out the fields as listed below.

   Name: UFT_Jenkins
   Framework: UFT
   CI Server: Jenkins CI
   SCM Type: Git
   Repository: http://nimbusserver:8091/git/UFT_Tests.git
   User name: root
   Password: Password1



7. Click the **TEST CONNECTION** button to make sure the configuration is correct.
   If everything's correct you'll see Connection succeeded

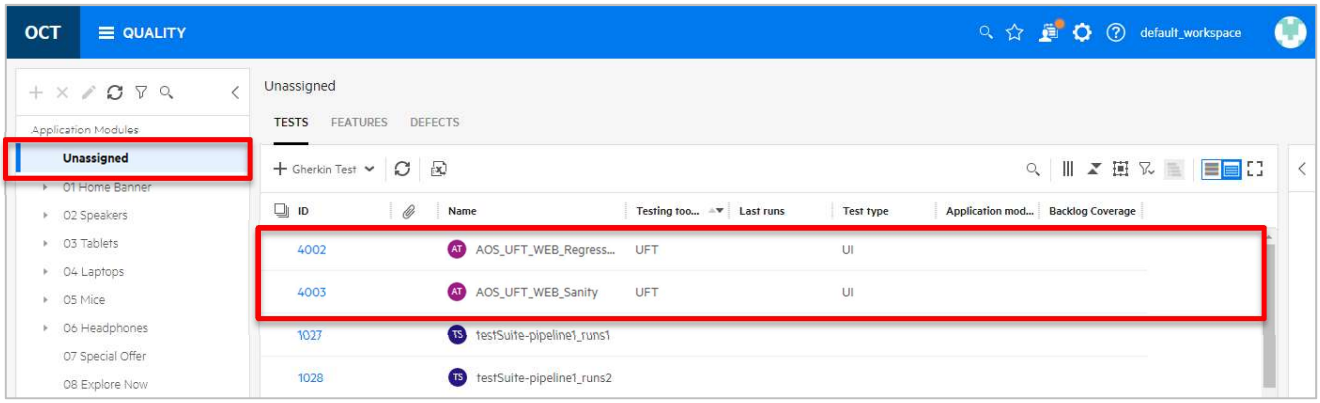
8. Click **Save** to complete the connection.
   ALM Octane creates a Jenkins job that connects to the repository and discovers the UFT One tests and data tables. You can see this job in Jenkins if you open the **Test** tab.

   The test scripts and the data table content are available in UFT only.
   If you add a new test in UFT or change an existing one, the changes are reflected in ALM Octane.

9. In Octane, select the gear icon and choose **Return to main application**.

10. In Octane, select the **Quality** module -> **Tests**

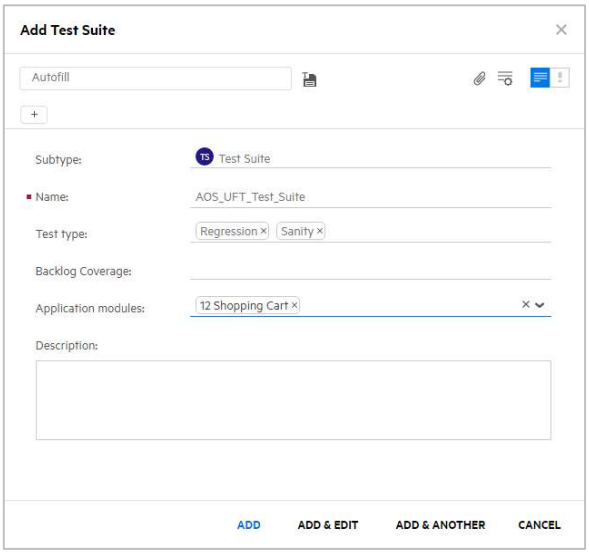
11. In the **Applications Modules** panel on the left, select the **Unassigned** option.
    This will display the two UFT tests that we placed in the Git repository (refresh if they don't show).
    These tests are currently unassigned until we edit and assign them to an application module.



12. Next we need to create a **Test Suite** in Octane.
    A Test Suite is a collection of tests to be run as a group or in a pipeline.
    This is similar to a **Test Set** in ALM QC.

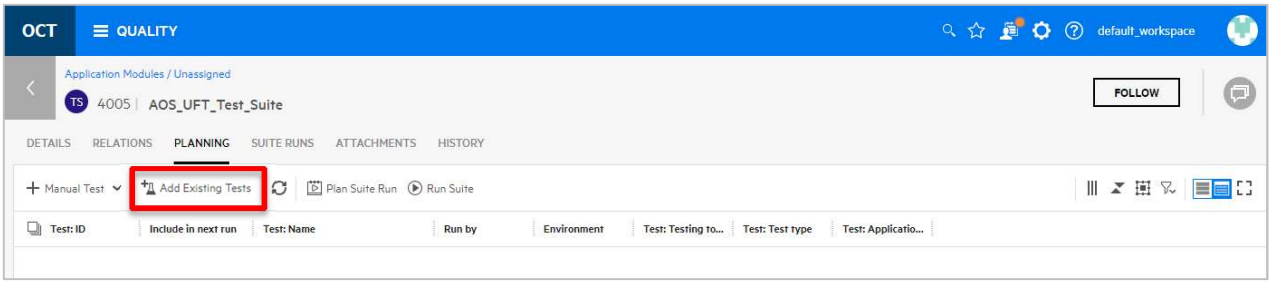
    Select the **+** option and fill out the form as shown below.

Subtype:	Test Suite
Name:	AOS_UFT_Test_Suite
Test type:	Regression + Sanity
Application model:	12 Shopping Cart

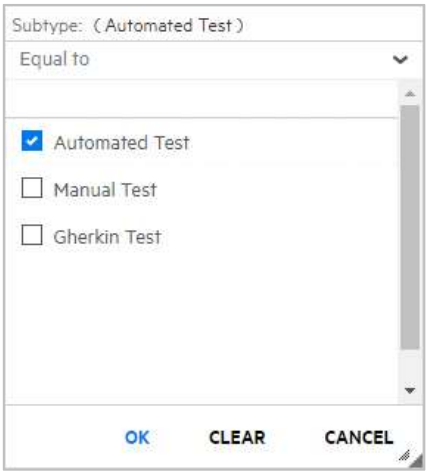


13. Select **Add & Edit**.

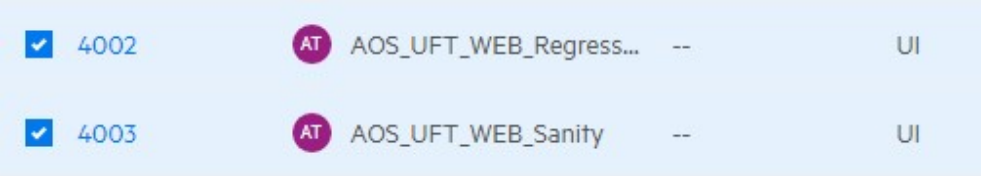
14. Select the Add Existing **Tests** option and click the test flask icon on the left to add tests.



15. Select the **+ Add Filter** option and select **Subtype : Equal to : Automated Test** to select UFT One tests.

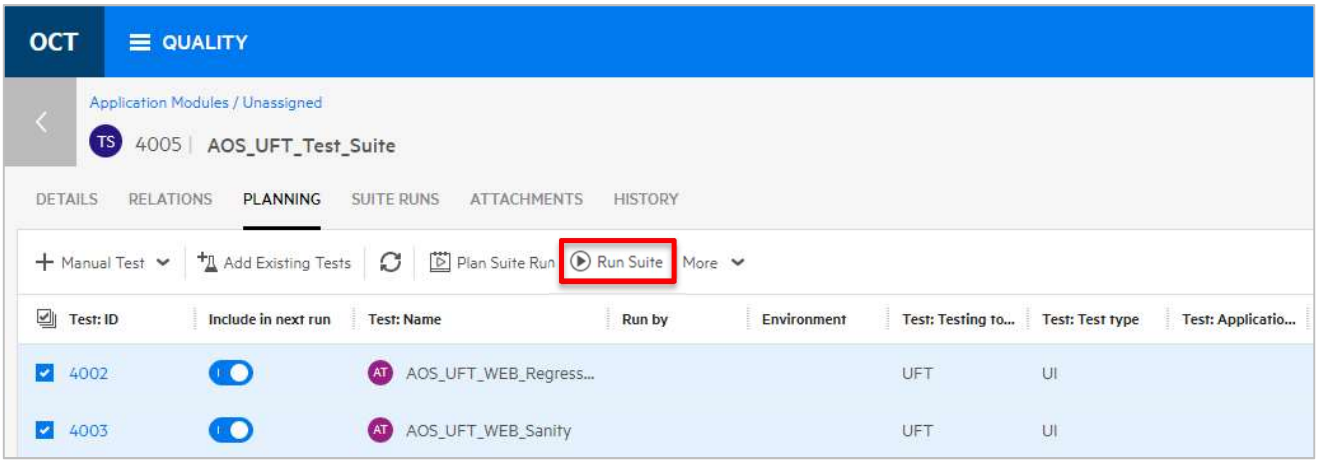


16. Select both tests and click **Add & Close** to add them to the Test Suite.



You've now added these two test to the Test Suite we created. When you run this test suite, both tests will run in the order shown, top to bottom. You can also drag and drop tests to reorder them or turn on or off specific tests.

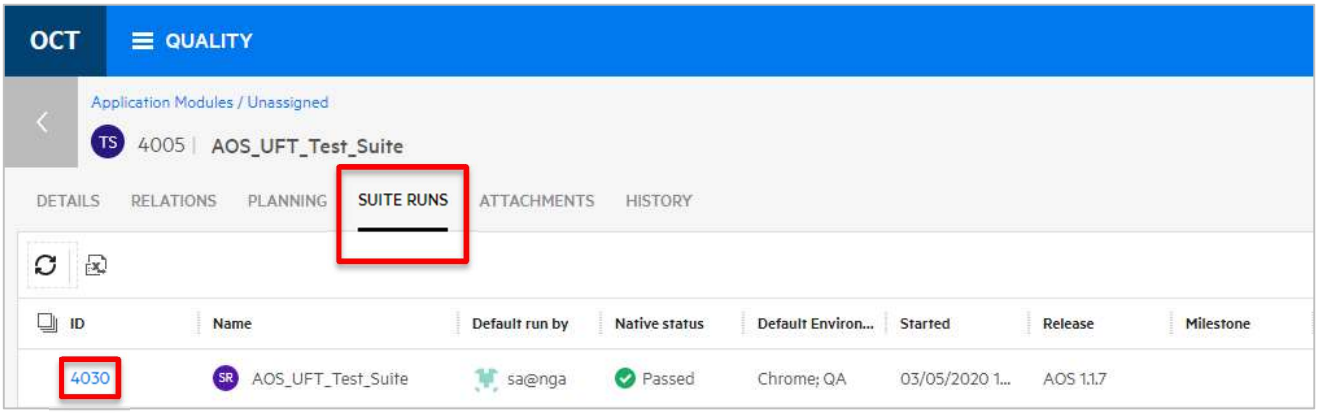
17. Click the option to **Run Suite** to run the entire test suite.



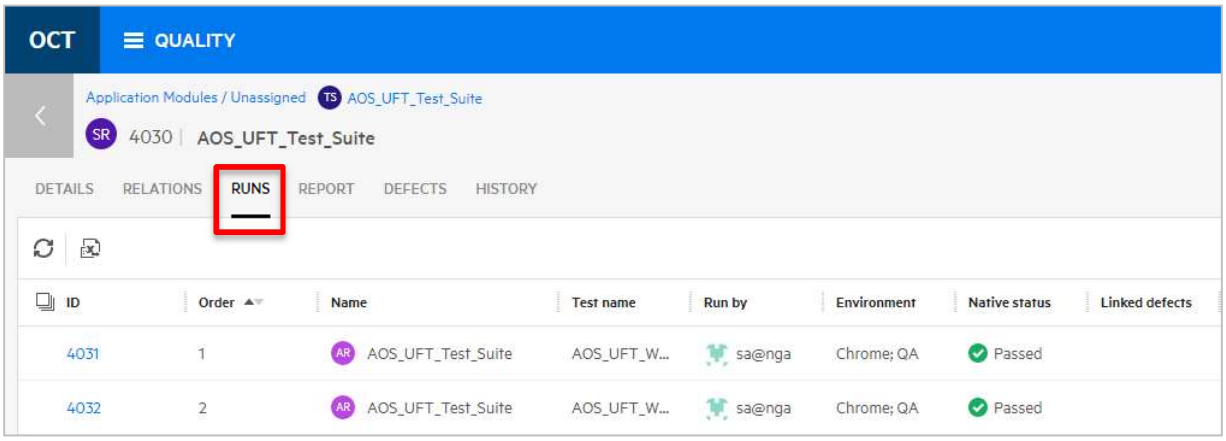
18. In the **Run AOS_UFT_Test_Suite** dialog, leave all the defaults and click **Let's Run!**

19. If you look at the **Jenkins** tab you should see a Jenkins job starting to run.
    Eventually you'll see each test opening a tab in your browser and running through their test steps.

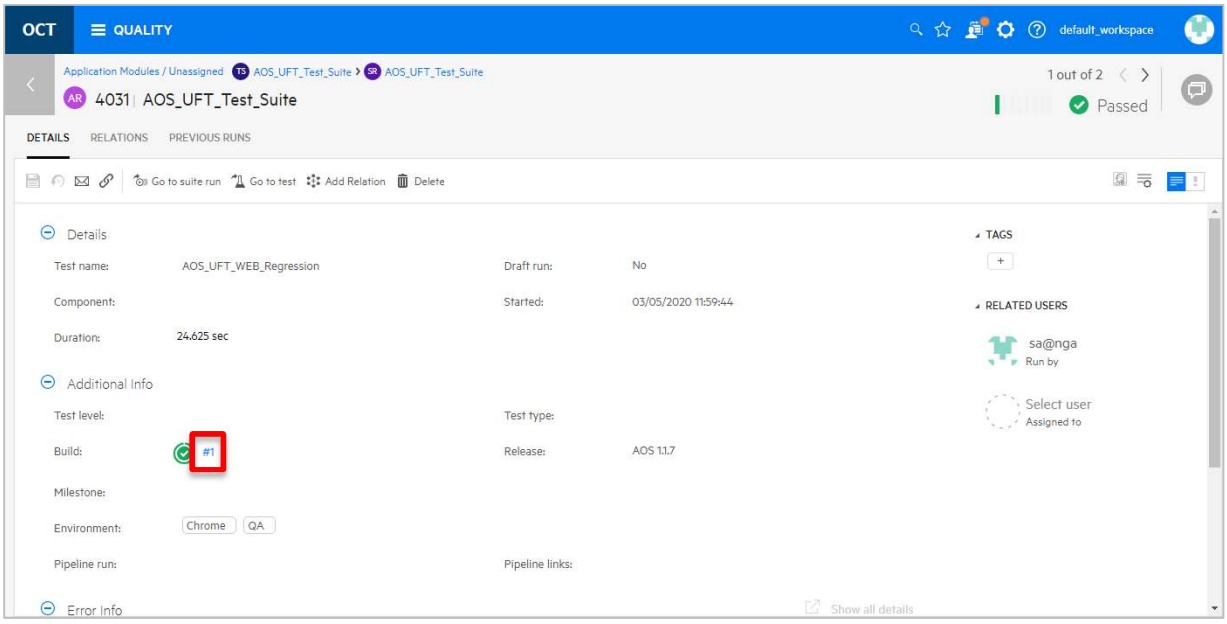
20. After the tests have finished running, wait a few seconds and then select the **Suite Runs** tab.
    You should see that the Test Suite's status now displays **Passed**.



21. Select the **numeric link** for the test suite and then select the **RUNS** tab.
    You should see both test run statuses.



22. Select one of the test's numeric link to drill down into the results of your automated run (AR). From here you can go to the test, report a defect or drill into the Jenkins run results.



23. Select the build number (e.g. **#1** in the image above) and **Accept** any redirect to Jenkins.

24. From Jenkins, click on the **UFT Report** link  UFT Report and then click on the report name link to see the actual UFT One run results details.

Type	Report name	Timestamp	Status	Folder
	AOS_UFT_WEB_Regression[1]	05/03/2020 12:00:03		Open
	AOS_UFT_WEB_Sanity[1]	05/03/2020 12:00:28		Open

Exercise 15: Integrating Security Testing

ROLE: INTEGRATION OR AUTOMATION ENGINEER

Security testing is often left as the last step in a DevOps process. That's not right. Security, like all testing should be continuously checked for issues or defects. Fortify offers multiple options that can scan your code and website and produce a comprehensive report.

In this exercise we'll scan part of the AOS source code (scanning the full 500,000 lines takes over 10 hours) and produce a report (a Fortify Project Report or .fpr file) which can be viewed with the **Fortify Audit Workbench** (AWB). This tool is really an Eclipse IDE in disguise and FPR files are Eclipse projects. Since understanding the relationships and dependencies between code modules is critical to finding errors, it makes sense to use an established IDE engine to also display security issues.

Running a Fortify Security Scan

1. We need to have the following VM's running for this exercise:
   - NimbusServer

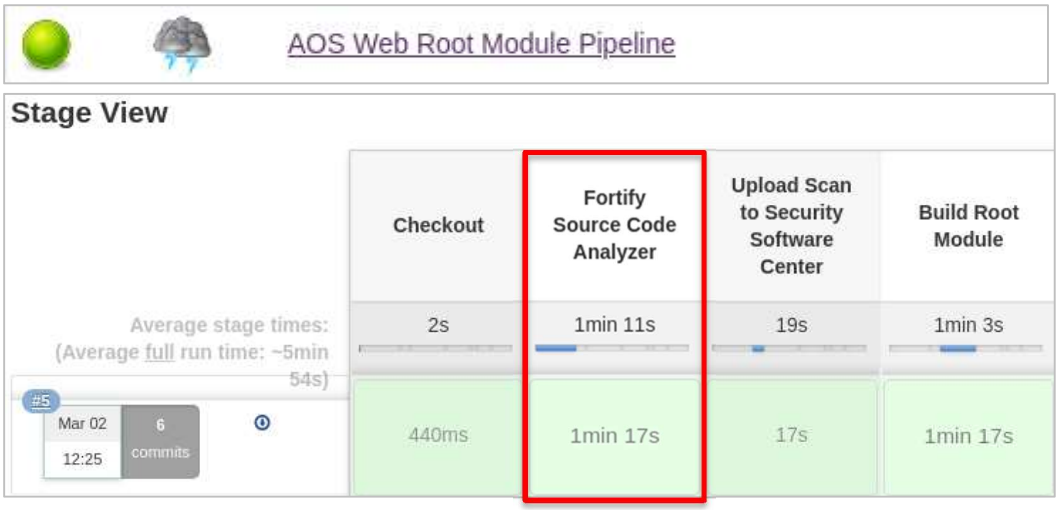
2. Run the following commands or verify that these containers are already running. Allow enough time for these containers to settle down (5-8 minutes or so).

   ```
   $ docker login        (Your instructor will provide the credentials)
   $ nimbusapp sca:19.1.2 start
   $ nimbusapp ssc:19.1.0 start
   $ nimbusapp aos:2.2 start
   $ nimbusapp devops:2.2.1 start
   ```

3. From **NimbusServer**, start Chrome and open the **Jenkins** shortcut.

4. Verify that from the Jenkins home page that **sca** shows as a connected, idle node.

5. From the Jenkins home page, **run** the **AOS_Web_Root_Module_Pipeline** and verify that the Fortify Source Code Analyzer stage ran successfully. This can take a few minutes to finish.



As the result of a code scan, Fortify produces an **FPR** file (Fortify Project Report).
Since we configured the scan for the root module in the AOS code it produced a **root.fpr** file.
That file was also uploaded to Fortify SSC (Software Security Center) in the next stage.

Using the Fortify Audit Workbench

Next, we'll copy the file to the Audit Workbench (AWB) container so we can view the vulnerabilities that it produced. AWB can open this file but it can also be viewed in SSC's scan database.

1. Start the auditworkbench Docker container by running the following command:

```
$ nimbusapp auditworkbench:19.1.2 start
```

2. If prompted to check for updated rules, select **Yes**.
Fortify's security team regularly updates the rules it uses to scan code for the latest, discovered code vulnerabilities so it's important to always use the latest rules.

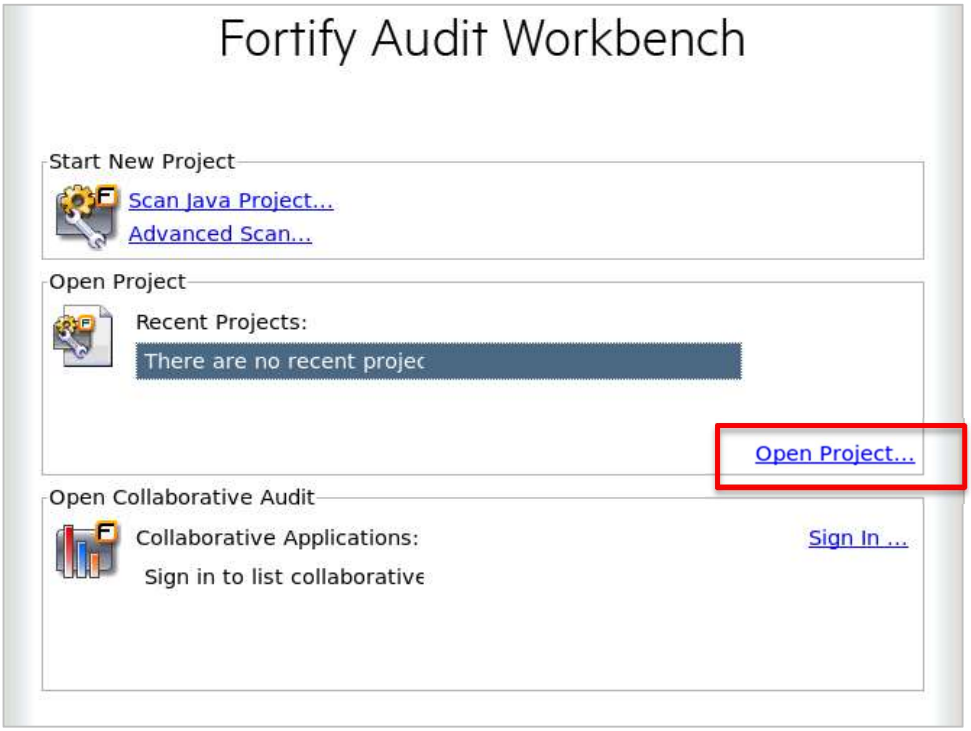
3. We can't immediately view this file so we'll copy it to somewhere else where we can look at it.
Open a terminal on NimbusServer and type the following two commands:

```
$ docker cp sca:/var/jenkins/workspace/aos-web-root-module-pipeline/root.fpr .
$ docker cp ./root.fpr auditworkbench:/root/root.fpr
```

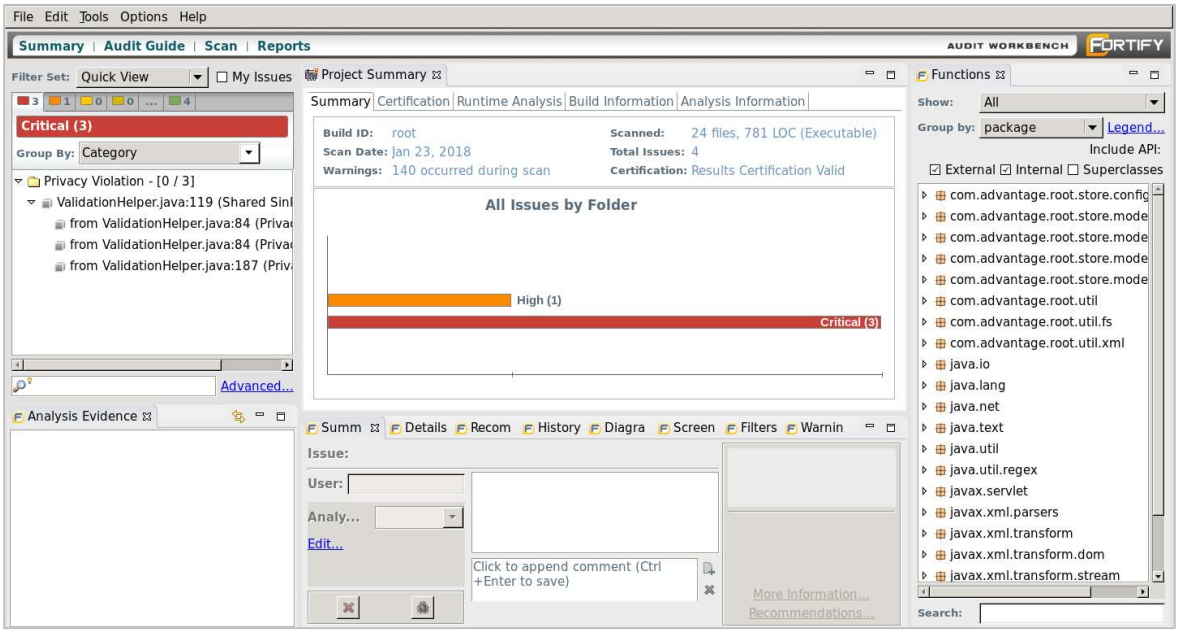
We're moving the file to the auditworkbench container because that container has our FPR viewer application called Audit Workbench (AWB). Docker's copy command (cp) is a nice way to move files between the different container file systems.

The Fortify Audit Workbench is actually a modified version of the Eclipse IDE.

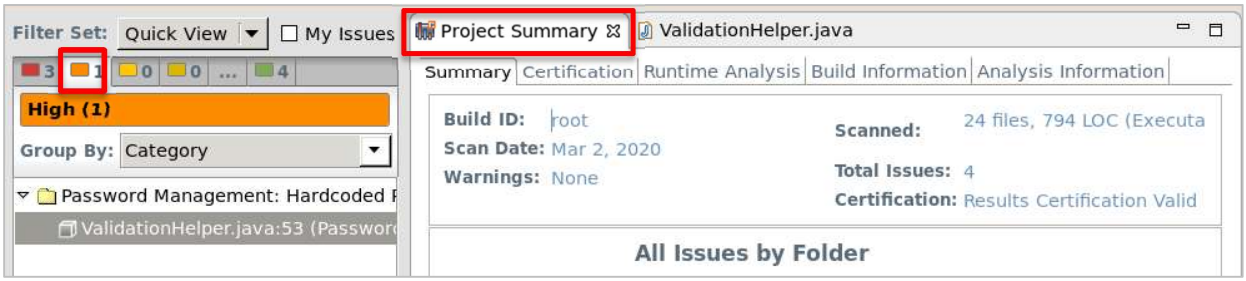
4. In the AWB window select **Open Project…**



5. Browse to the root directory (+ Other Locations - **/**) and select the **root.fpr** file to open.
6. When **AWB** finishes loading you'll see **1 High error** and **3 Critical errors**.
   Open up the tree view on the left.



7. Select the **High** filter in the top left menu and examine the error.
   The scan found a hard-coded password pattern in the code.



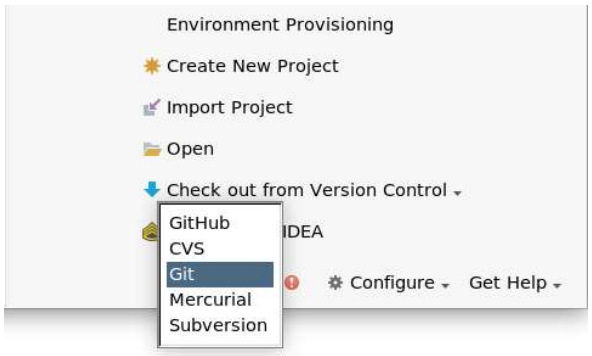
   You can also find the following scan statistics for the root module in the **Project Summary**:
   - 24 files
   - 794 Lines of Code (LOC)
   - 4 total issues

8. **Browse** around in the interface and see what you can find.
   Previously we did scan the whole AOS source code and its libraries but it took a few hours (it's over 500,000 LOC). The complete source code is on the DevOps container.

9. Run the following command to start Intellij:

   ```
   $ nimbusapp intellij:2.2.0 start
   ```

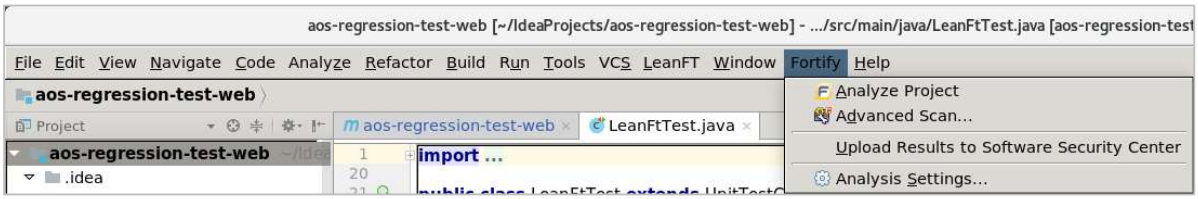
10. Select to Checkout from version control -> Git



    From the Git Repository URL select:

    aos-regression-test-web

11. Once IntelliJ has finished loading, select the Fortify menu at the top:



There are four options listed:

- Analyze Project
- Advanced Scan…
- Upload Results to Software Security Center
- Analysis Settings

12. Select Analyze Project and let it run the scan.
    When prompted for the login credentials for SSC enter:

    - Username: **admin**
    - Password: **Password1**

13. After the scan is finished you can open Software Security Center, login and examine the results.

14. **Close** the **Audit Workbench** UI and the **Intellij** UI when you're done browsing through them.

Using the Fortify Software Security Center

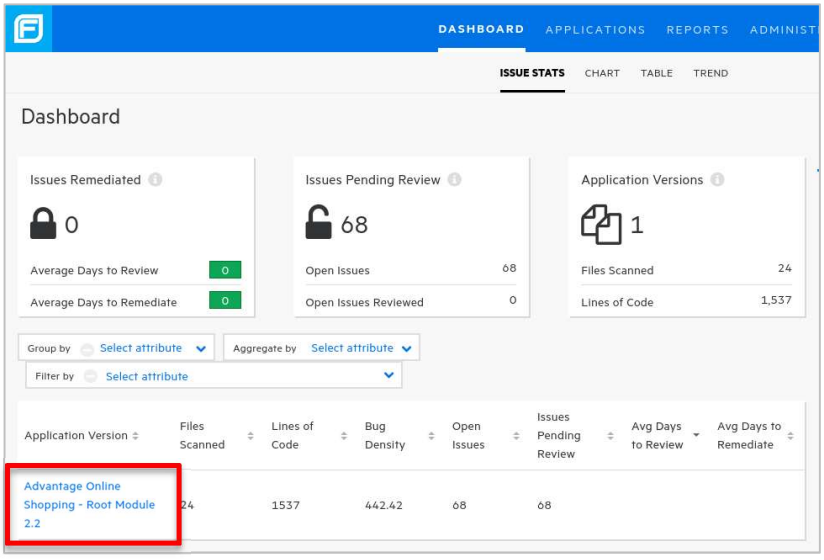
The Fortify Software Security Center (SSC) is a web-based application that houses the results from security scans and consolidates and aggregates the data into actionable tasks.

1. We need to have the following VM's running for this exercise:
   - NimbusServer
2. Run the following command or verify that this containers is already running:

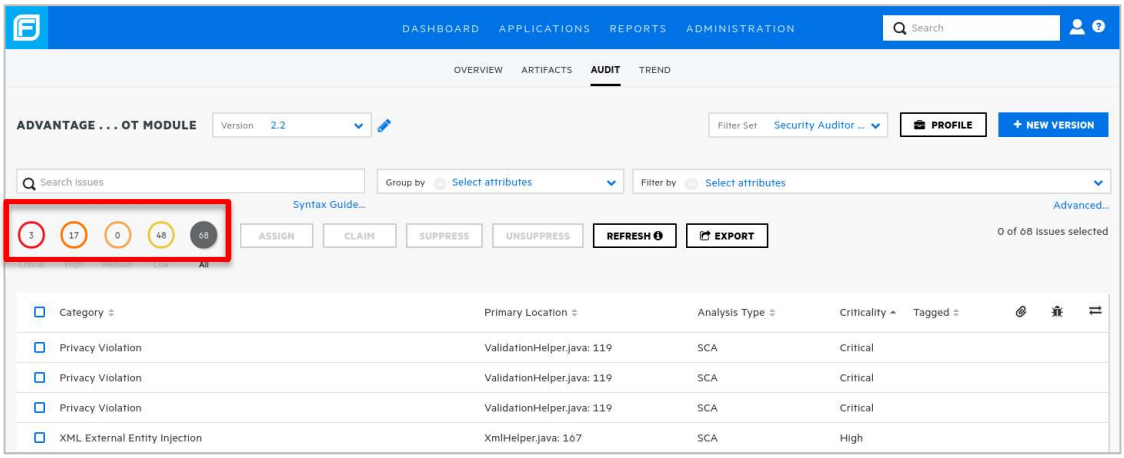
```
$ nimbusapp ssc:19.1.0 start
```

3. **Open** your browser to the **Fortify Software Security Center (SSC)** shortcut.
4. **Login** with these credentials:
   Username: **admin**
   Password: **Password1**

5. Once logged in you'll see the dashboard.
   In this SCA container we've pre-imported the **root.fpr** scan file from the previous exercise.



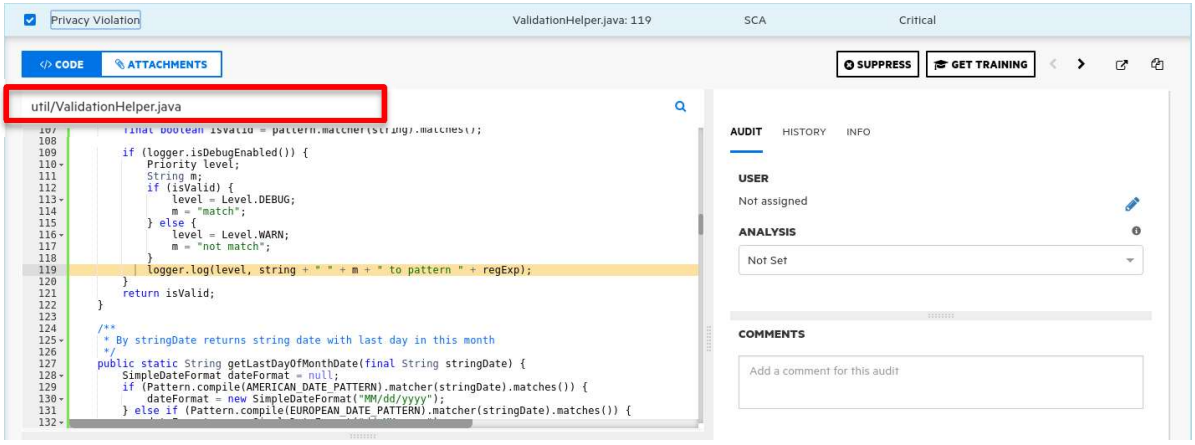
6. **Examine** some of the **Issue Stats** and then **click** on the **Advantage Online Shopping – Root Module 2.2** link. This bring you into the **Audit** section of SSC.



Some of the data you can find on this screen:

Total Issues:	68
Critical Priority Issues:	3
High Priority Issues:	17
Medium Priority Issues:	0
Low Priority Issues:	48

7. In the **Search Issues** box at the upper left enter **Password** and hit **Enter**.
8. Click on the **Privacy Violation** text to view the source code.



9. The item shows the same file **(ValidationHelper.java)** that we saw directly in Audit WorkBench. The next step in a security scan is to either suppress the issue or assign it to someone to be resolved.

10. Browse around this interface and see what else you can find.

Integrating Fortify results in Octane and the Jenkins CI

This exercise describes the process needed to integrate the Fortify results from SSC and viewing them in Octane.

Basically, the vulnerabilities data will only appear when NEW code vulnerabilities are found. So, the first time you run a scan and it is populated into SSC, the data will come into Octane. All future scans will not produce any vulnerabilities and thus no new data. So, you can only see the initial vulnerabilities with the first pipeline scan that populates into SSC. However, if you change the code and introduce new vulnerabilities, they will show up (but not the original ones, just the changed code ones).

1. We need to have the following VM's running for this exercise:
   - NimbusServer with at least **26 GB** of available memory (do not attempt with less)

2. Run the following commands which will start the devops and AOS containers.

```
$ nimbusapp devops:2.2.1 start
$ nimbusapp aos:2.2 start
```

3. Run the following commands which will reset the SSC container to its initial condition (empty). Select **Y** (Yes) to delete the container when prompted and wait about 4 minutes.

```
$ nimbusapp ssc:19.1.0 down
$ nimbusapp ssc:19.1.0 up
```

4. Open **Octane** to the **Pipelines** module and **delete** the aos-web-root-module-pipeline pipeline by selecting the three dots and **View** and then the trashcan icon.
   We'll recreate

5. Create a new pipeline

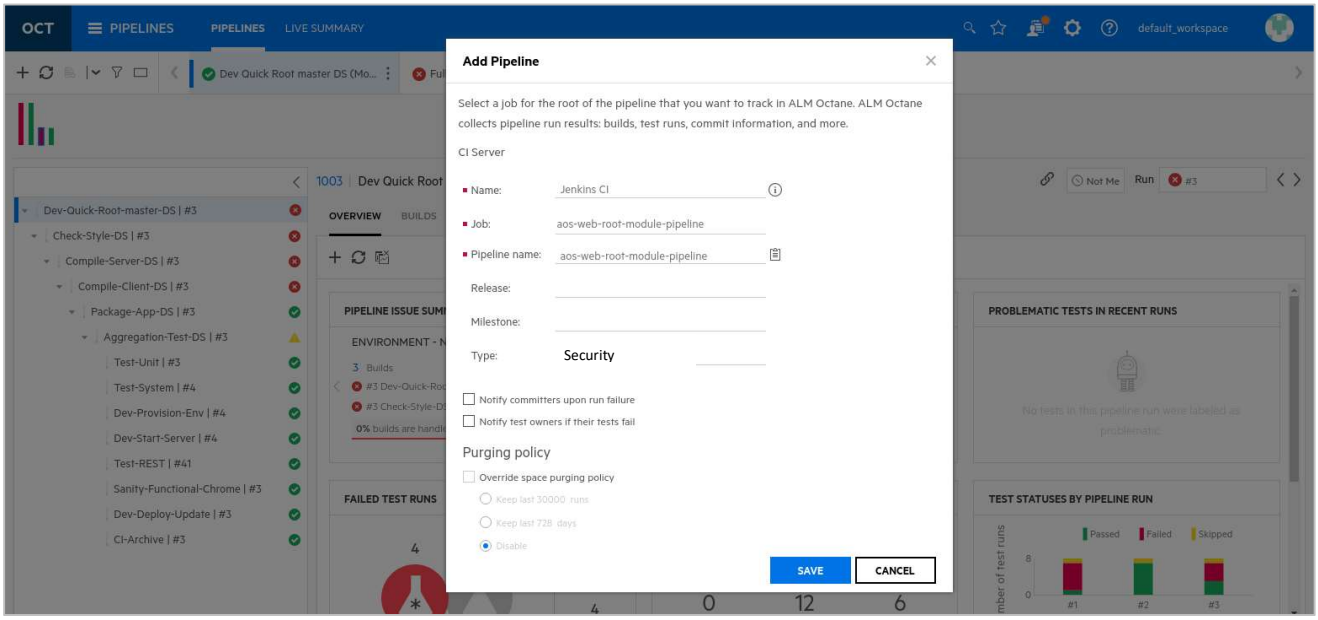
6. Run the following command which will start the IntelliJ container and the UFT Dev agent.

```
$ nimbusapp intellij:2.2.0 start
```

7. In the desktop **System Monitor** wait for the CPU activity to settle down.

8. **Login** to ALM Octane (sa@nga, Password1) and navigate to the **Pipelines** module.

9. Create a new pipeline (+) based on **AOS_Web_Root_Module_Pipeline** and select the **Type** checkbox for **security** before clicking **SAVE** (you must do this).



10. Ok – Now we should have everything in place.

    a. SSC should not have a current application nor results showing in it.
    b. Octane should have a pipeline defined (AOS_Web_Root_Module_Pipeline) but it hasn't been run as an Octane pipeline yet.
    c. Jenkins has all the configured connections which have been tested.
    d. The UFT Dev agent in IntelliJ is still active and it hasn't been more than four hours since it started.

11. Double-check (using **`$ docker ps`**) that all nine of these containers are running:

autopass	**aos_main**	**intellij**
devops	**aos_postgres**	**sca**
octane	**aos_accountservice**	**ssc**

12. Check that you still have at least **2.4 GB** of memory **available** on NimbusServer by running the **System Monitor** on your desktop.

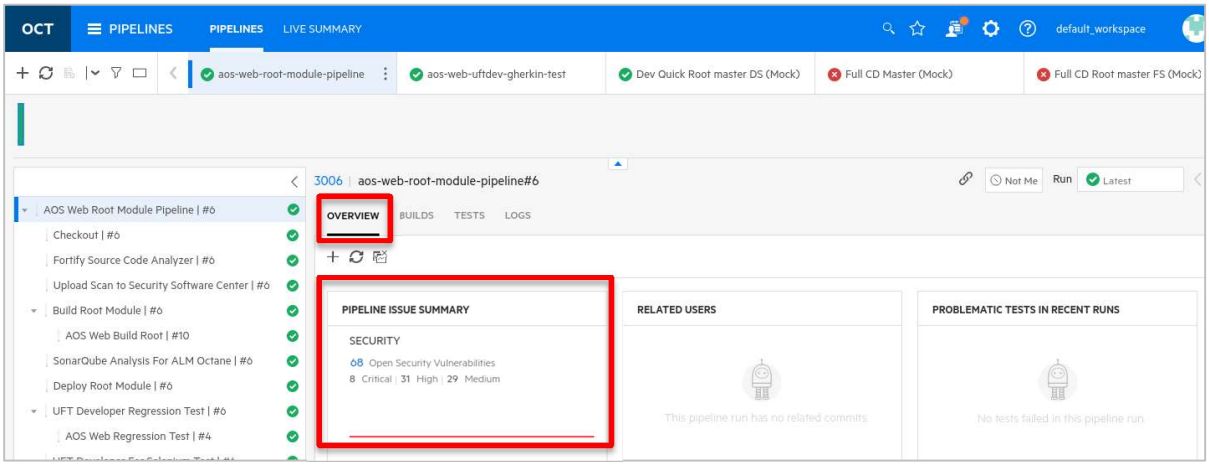
13. In Jenkins configuration, verify that the **ALM Octane CI** plugin shows that the **Test Connection** succeeds. If not, run the utility job called **Update Octane API** and check it again.

14. In either Jenkins or Octane run/build the **AOS_Web_Root_Module_Pipeline** job.

15. Watch the progress in Jenkins. If the job fails, find out why and correct it – you'll also have to delete/recreate the Octane pipeline again.

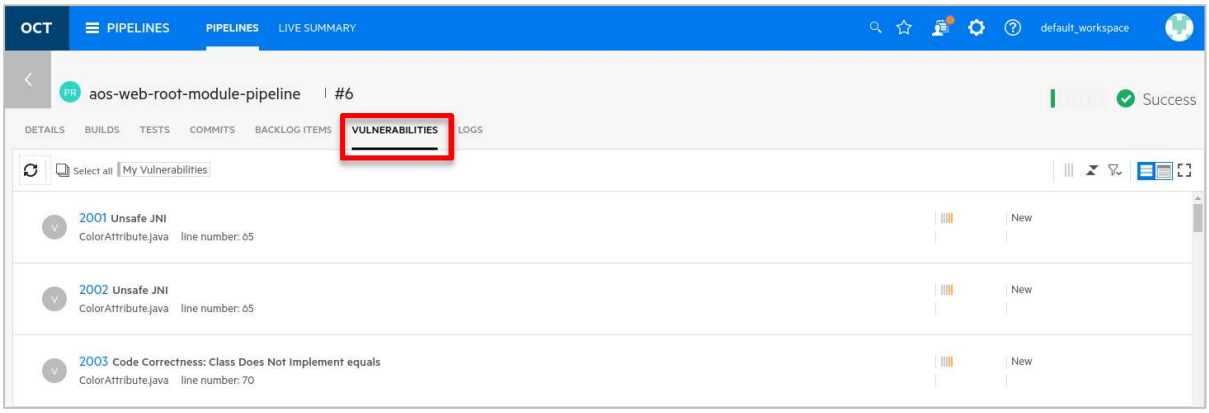
16. When the job finishes, wait about 4 minutes before bringing up Octane and **refreshing** the Pipelines module. Initially it may appear that the Security data isn't showing up. Just refresh your browser.

17. You should see something like the following:



18. Select the pipeline run ID link and then select the **Vulnerabilities** tab.
    You can also get there by clicking on the 68 Open Security Vulnerabilities link

19. You should see the data similar to below.

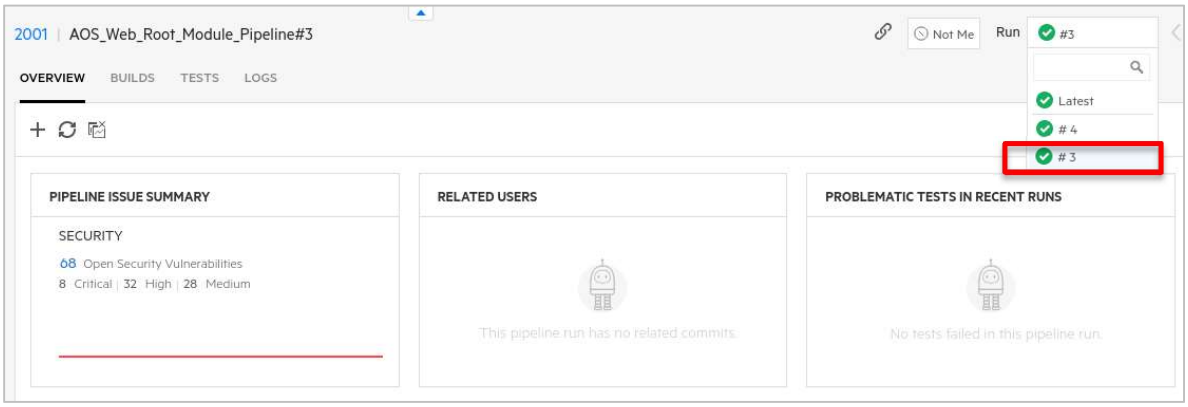


20. Pat yourself on the back for correctly configuring this integration!

    NOTE: If you re-run the **AOS_Web_Root_Module_Pipeline** again, you won't see the vulnerabilities appear again. This is because no **NEW** vulnerabilities were detected. If you want to see your original vulnerabilities again, just click on the Run pulldown and find the previous run where you did get data from the Fortify scan (see image below).

If you changed the code to add new vulnerabilities, then only those new vulnerabilities would appear.

If you want to see the vulnerabilities scan of a **previous** pipeline run, select that run from the dropdown on the right as shown below:



Also note that if you delete the pipeline from Octane and re-add it, if there is still vulnerability data from a previous run, it won't appear again in Octane (unless new vulnerabilities are published to SSC). The only way to get this data back into Octane is to remove both the pipeline in Octane and the scan artifact in SSC and then set the integration up again.

You can remove the scan artifact in SSC by selecting the artifact application link and then selecting Profile. From the Profile window, select Application Settings and then you can delete the artifact by clicking on the DELETE button.

Nicely Done!

Running WebInspect Dynamic Scans

This exercise describes how to run a WebInspect dynamic scan on AOS and upload those results to Fortify SSC.

Basically, the vulnerabilities data will only appear when NEW code vulnerabilities are found. So, the first time you run a scan and it is populated into SSC, the data will come into Octane. All future scans will not produce any vulnerabilities and thus no new data. So, you can only see the initial vulnerabilities with the first pipeline scan that populates into SSC. However, if you change the code and introduce new vulnerabilities, they will show up (but not the original ones, just the changed code ones).

1. We need to have the following VM's running for this exercise:
   - NimbusServer
   - NimbusClient

2. Run the following commands or verify that these containers are already running.
   Allow enough time for each of these containers to settle down (4-6 minutes or so).

   ```
   $ nimbusapp ssc:19.1.0 start
   $ nimbusapp aos:2.2 start
   $ nimbusapp devops:2.2.1 start
   ```

3. In your Chrome browser on NimbusServer select the **Jenkins** browser shortcut.

4. On the left edge, select the **New item** option and then click **Freestyle Project**.

5. Name the new item to be **AOS_Web_Dynamic_Scan**

6. Set the description to be "**This job runs a WebInspect scan on the local Nimbus-based AOS website.**"

7. Set the value "Restrict where this project can be run" to be on **nimbusclient_uft**

8. Add the following code to the **Build** section pulldown as **Execute Windows Batch Command**.
   This command will run a quick (~7 minutes) vulnerability scan on the AOS website and produce a couple of reports.

   ```
   C:\Progra~1\Fortify\Fortif~1\wi.exe -u
   "http://nimbusserver.aos.com:8000/#" -ps 1 -s
   "C:\ProgramData\HP\HPWebI~1\Settings\Default.xml" -ep
   $WORKSPACE\SecScan\AOS_CmdScan1.fpr -r "Vulnerability" -y "Standard" -
   f $WORKSPACE\SecScan\AOSAppVuln.pdf -gp
   ```

9. Click **Save** to save your changes. This will create a new job called **AOS_Web_Dynamic_Scan** in the **All** tab in Jenkins.

10. In Jenkins, select to **Build** the **AOS_Web_Dynamic_Scan** job.

11. Switch to the **NimbusClient** image and if a security warning about a certificate pops up, click **OK**.

12. Click the flashing ball in Jenkins to view the log of the run. It should take about 6-10 minutes to finish. Ignore any warnings about addons.xpi that may appear (we're investigating this).

13. Once finished you can see the results in Jenkins or in WebInspect.
    To view it in Jenkins, navigate to $WORKSPACE/SecScan and view the PDF report.

Challenge Exercise #2

Copy the FPR file that was produced and upload it into SSC. Show the results to your instructor.

Exercise 16: Synchronizing with MF Connect

Configuring the Jira Container

Jira is a common Agile management and defect tracking tool that many organizations use. Those organizations often want to sync their Jira data with ALM Octane. In this section we'll explore that integration.

Jira is available as a Docker container in the public Docker Hub and we've provided a slightly modified version of that public container provided by Captain Action Hank (cptactionhank/atlassian-jira-software) that we'll use for this exercise.

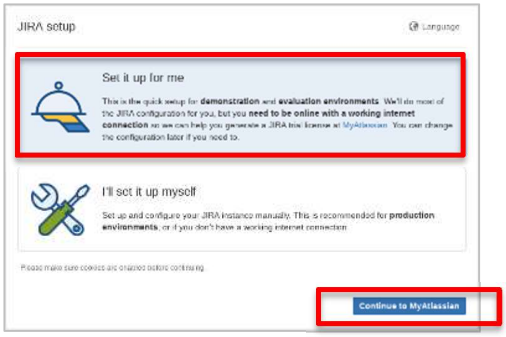
1. Start the following Docker containers for this exercise. (The AutoPass container also needs to be running, but it starts automatically with NimbusServer.)

```
$ nimbusapp octane:15.0.46.68 start
$ nimbusapp mfconnect:4.2 start
$ nimbusapp jira:8.1.0 start
```

2. From **Nimbus Server**, open a browser and navigate to Jira using the shortcut. The URL is http://nimbusserver.aos.com:8099.

3. If this is the first time Jira has been run, follow the directions below to configure it and obtain a license. Otherwise, skip to ***Creating a Jira Data Source***.

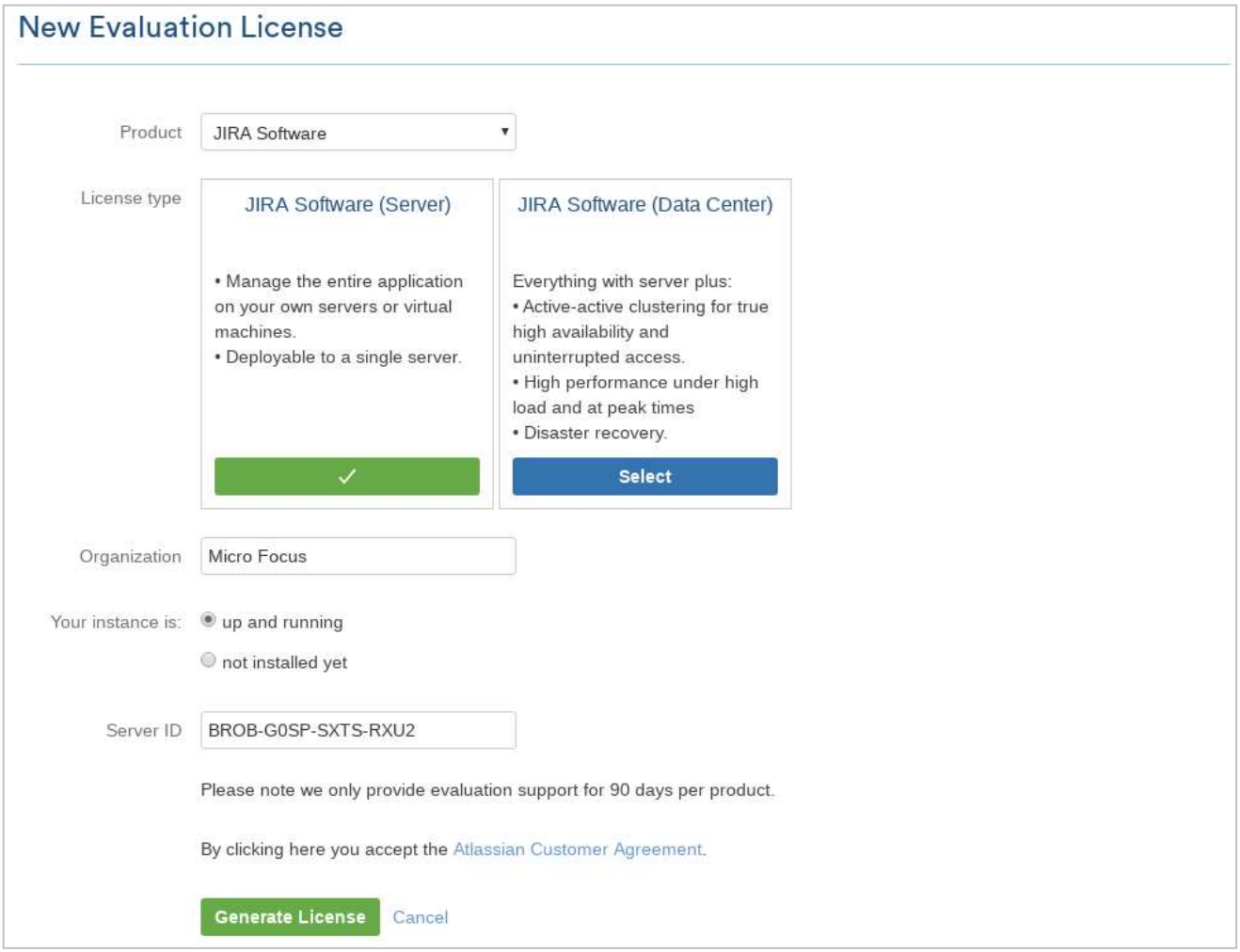
4. Select **Set it up for me** and click **Continue to MyAtlassian.**



5. Select **Sign up for an account** and enter the required fields.

6. You'll receive a confirmation email. Open it and click on the link to **Log In**.

7. On the New Evaluation License page, select the following:
   - Product: **Jira Software**
   - License Type: **Jira Software (Server)**
   - Organization: **Micro Focus (or your company)**
   - Your instance is: **Up and running**
   - Server ID: **<value is auto populated>**



8. Select **Generate License**
   This confirms your acceptance of the **Atlassian Customer Agreement** and provides evaluation support for 90 days.

9. In the confirmation box select **Yes**.

10. In the Administrator account setup section enter the following:
    - Email: **<your email account>**
    - Username: **admin**
    - Password: **Password1**
    - Retype password: **Password1**

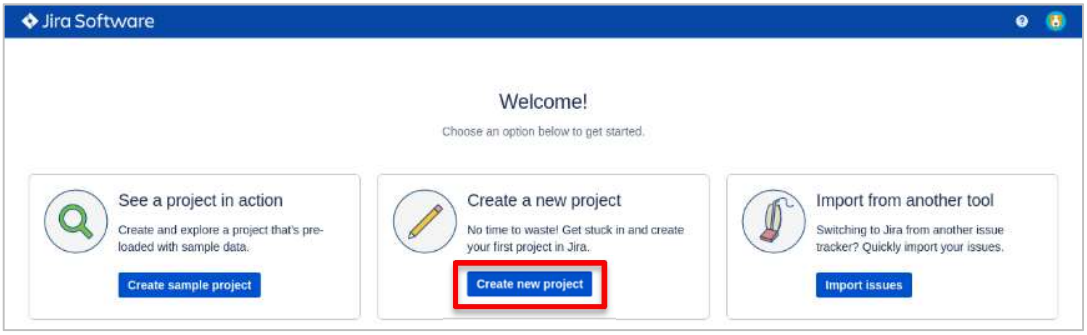
11. Click **Next**.

12. Wait while Jira finishes the setup.

13. After Jira finishes, click the **Let's get started** button.

14. Login to the **admin** account, choose your language and click **Continue**.

15. Optionally, choose an avatar and click **Next**. You will see the welcome page.



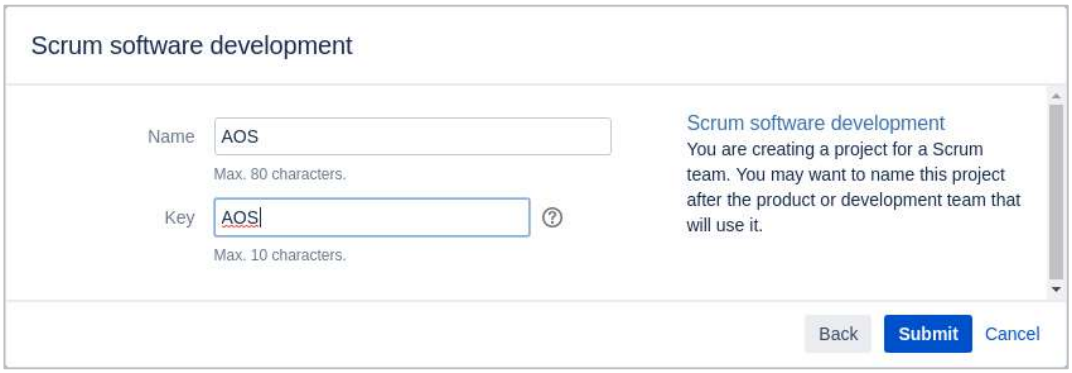
Creating a Jira Data Source

Before we can synchronize anything, we need some data and a project to synchronize.
We'll create those next.

1. Click the **Create new project** button.

2. Select the **Scrum Software Development** and click **Next** and then **Select**.

3. Set the name and key to be **AOS** and click **Submit**.

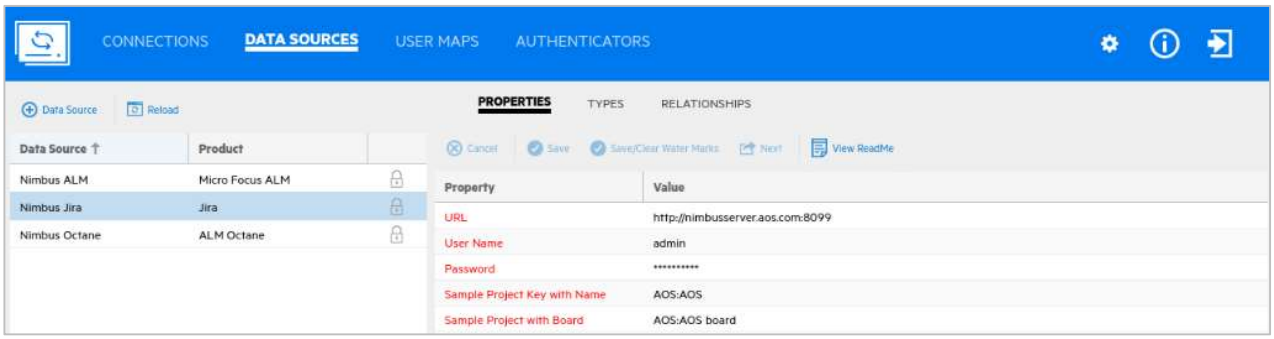


4. Open a new browser tab and navigate to MF Connect using the shortcut. The URL is http://nimbusserver:8081/ConnectWeb/

5. Login with these credentials (case sensitive):
   - Username: **Administrator**
   - Password: **Administrator**

6. In MF Connect, click the top right gear button and choose **Refresh Server Data**.

7. Click on the **DATA SOURCES** tab on the top and click on **Nimbus Jira** in the list on the left. On the right side, in the properties tab, you'll see the configuration that MF Connect will use for Jira.



Connecting the Octane API

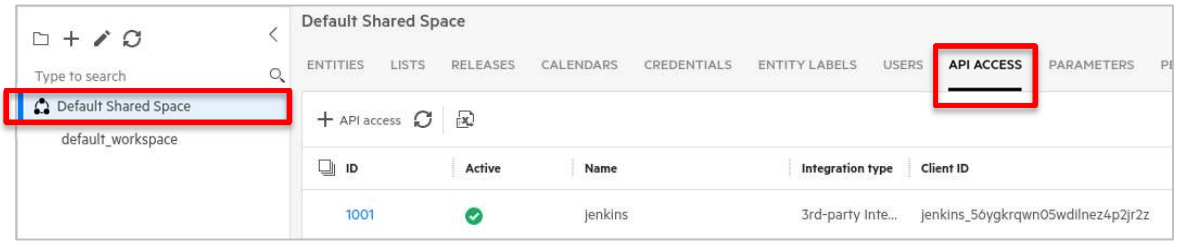
The first step in synchronizing ALM Octane and Jira is to make sure they can communicate via the API. Start by generating a Client ID and Client Secret and then copying those to MF Connect.

1. Open a new browser tab and login to Octane with:
   - Username: sa@nga
   - Password: Password1

2. Click on the Settings (gear) icon in the topmost tool bar and select **Spaces** under Administration.

3. On the left side select **Default Shared Space**.

4. In the tabs to the right select **API ACCESS**

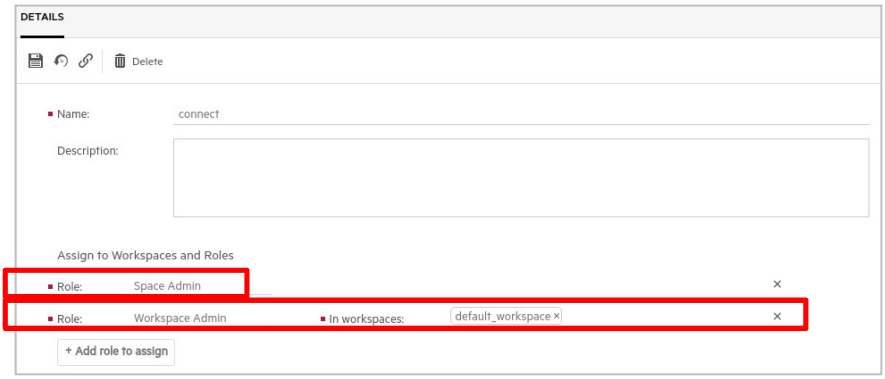


The **MF Connect** API access key is called **connect**.

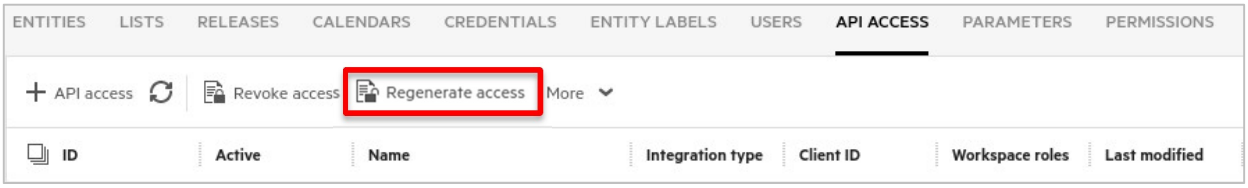
5. **Locate** the **connect** API access section and select the numeric link to edit it.

Verify the following roles from CI/CD and if needed, **Save** your changes:

Role: **Space Admin**
Role: **Workspace Admin** In workspaces: **default_workspace**



6. Select the **Go Back** arrow and then click **Regenerate access** and copy the new **Client ID** and **Client Secret**. You'll only see the new **Client Secret** in the popup.



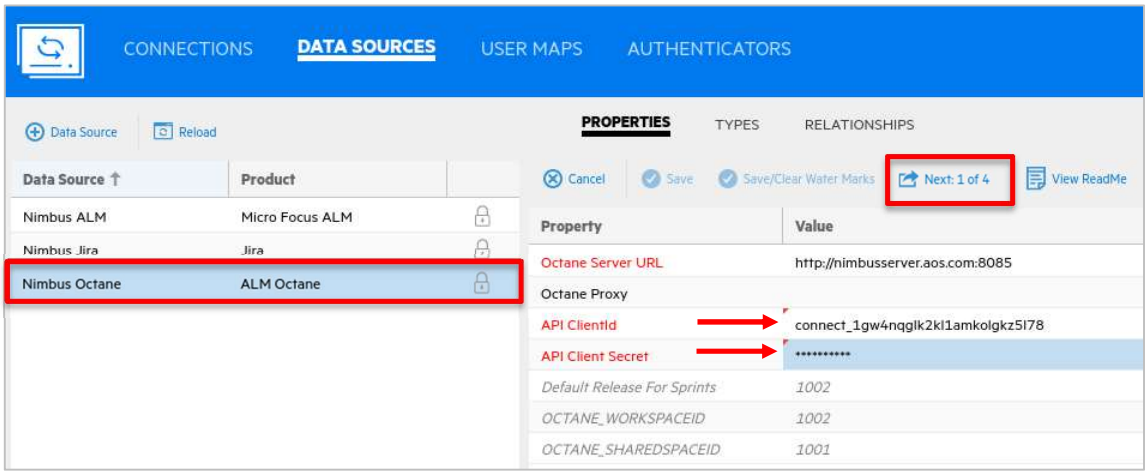
7. Open your terminal window and **paste** the content into it.
You'll get a couple of warnings but you'll have preserved the values for **Client ID** and **Client Secret**.

5. Click the **Settings** (gear) icon to return to the main Octane application.

6. On the **MF Connect** tab in your browser, select the **DATA SOURCES** tab.

7. Select the **Nimbus Octane** data source on the left panel. You'll probably get a validation error popup. Ignore this.

8. Paste in the new values for the **API ClientID** and **API Client Secret** on the right.



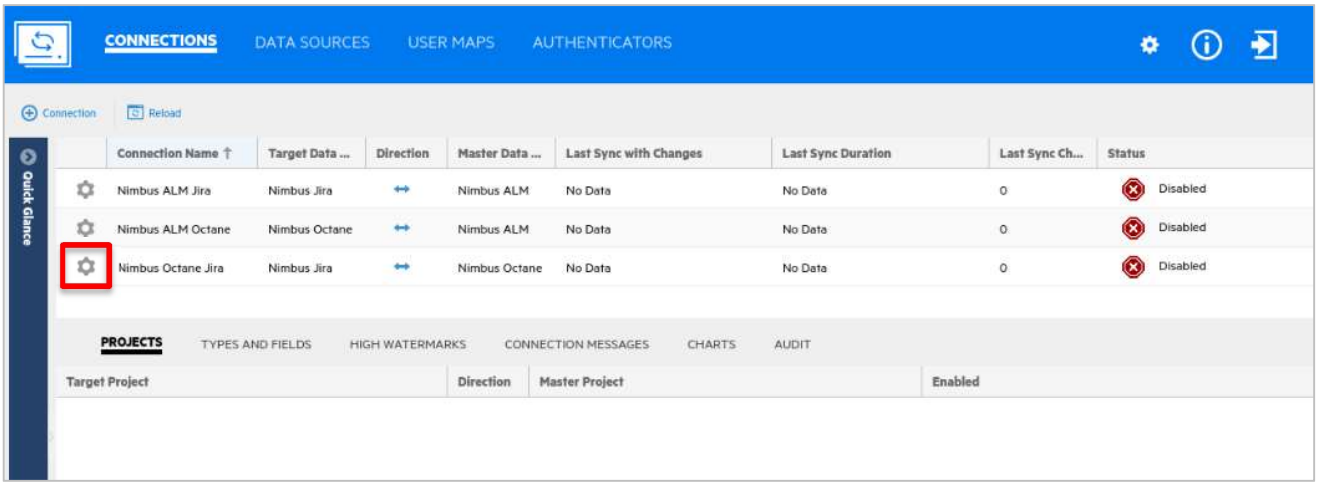
9. Click **Next 1 of 4** to continue. It will take a moment to load, but then it shows your saved changes. You've now configured the Octane connection.

10. While still on the **DATA SOURCES** tab, select the **View ReadMe** option on the right. This displays the latest details on the **Octane Connector**.

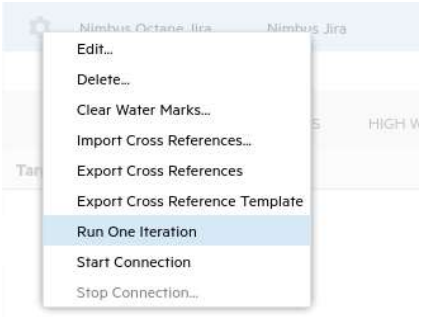
Synchronizing Jira and Octane

Now we're ready to sync data from ALM Octane and Jira. Since our Jira instance is empty, it will be the recipient of the selected Octane data. Only fields and links that have been mapped will sync to Jira. If you create new items in Jira at a later point in time, they will get synced to Octane since this is a bi-directional synchronization.

1. Select the **CONNECTIONS** tab at the top in **Connect**. We have three pre-configured connections in Connect:
    1. Nimbus ALM Jira – This syncs an ALM QC instance with Jira
    2. Nimbus ALM Octane – This syncs an ALM QC instance with Octane
    3. Nimbus Octane Jira – This syncs an Octane instance with Jira

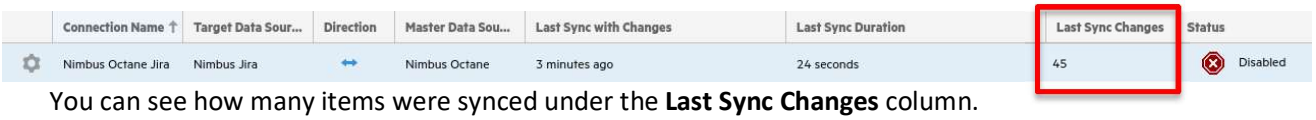


2. **Left click** on the gear icon to the left of the **Nimbus Octane Jira** connection. Select **Run One Iteration**.



This will run just one synchronization cycle. If you want it to run a continuous sync you would choose **Start Connection**.

3. The **Disabled** icon will turn green to show the synchronization cycle has started. It should take 30-90 seconds to finish the sync cycle.



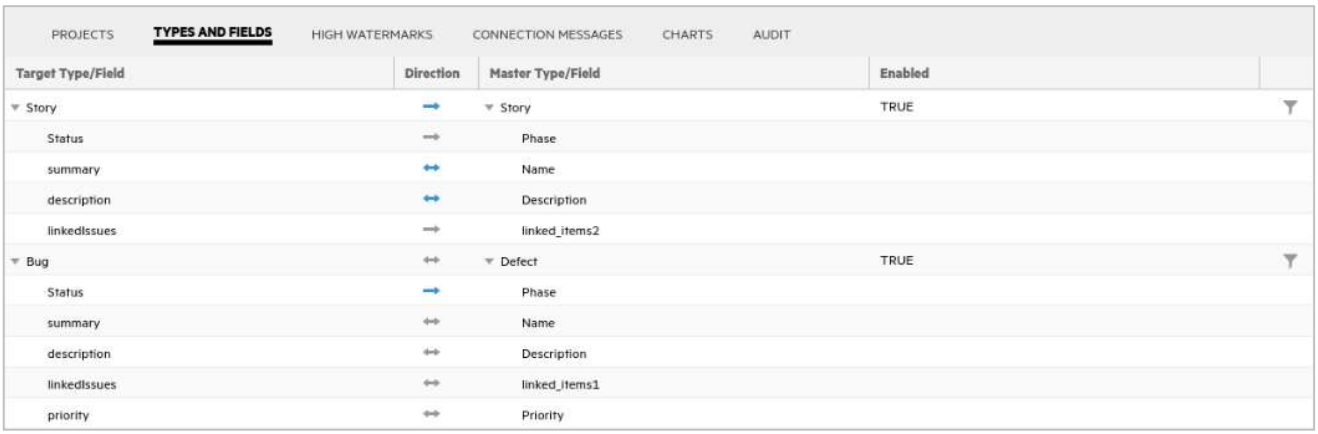
You can see how many items were synced under the **Last Sync Changes** column.

4. After a short time, click the **Reload** button to update your display. 

5. While you have the connection selected (it's blue), you can select the tabs in the bottom section to view the following information:

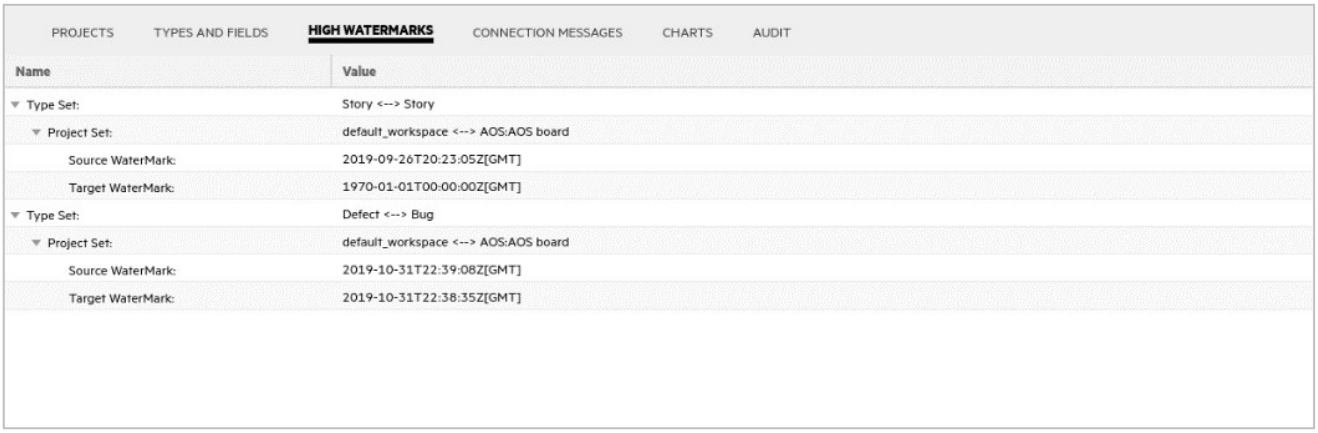
TYPES AND FIELDS

The **TYPES AND FIELDS** tab shows a list of the synchronized types and the fields that were synced. Each type can be expanded to show the associated fields and the direction of the sync.



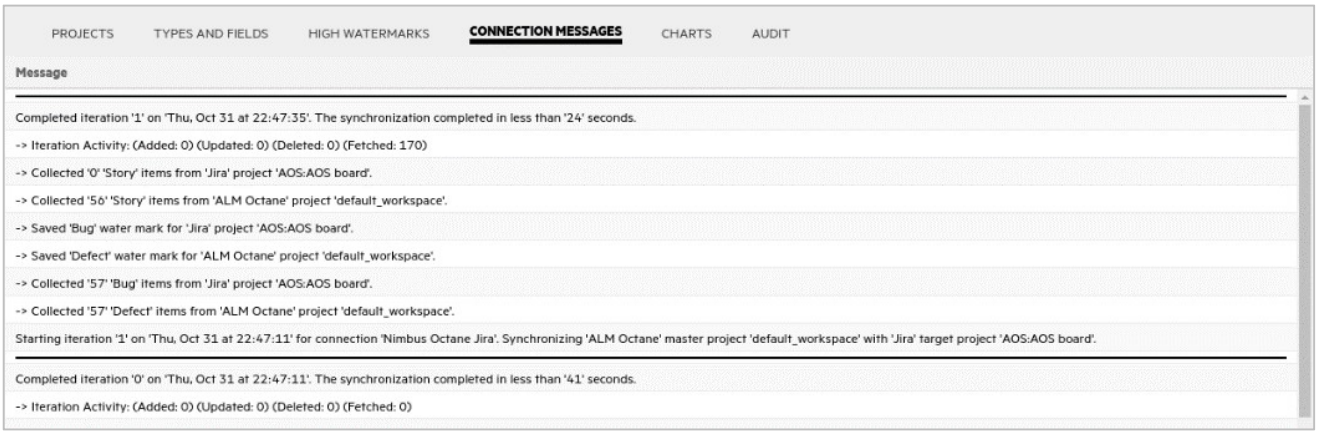
HIGH WATERMARK

High Watermarks allow MF Connect to know what has previously been synchronized. It acts as a time and place marker so that MF Connect compares what has been updated, added, deleted and so forth, after the High Watermark was made.



CONNECTION MESSAGES

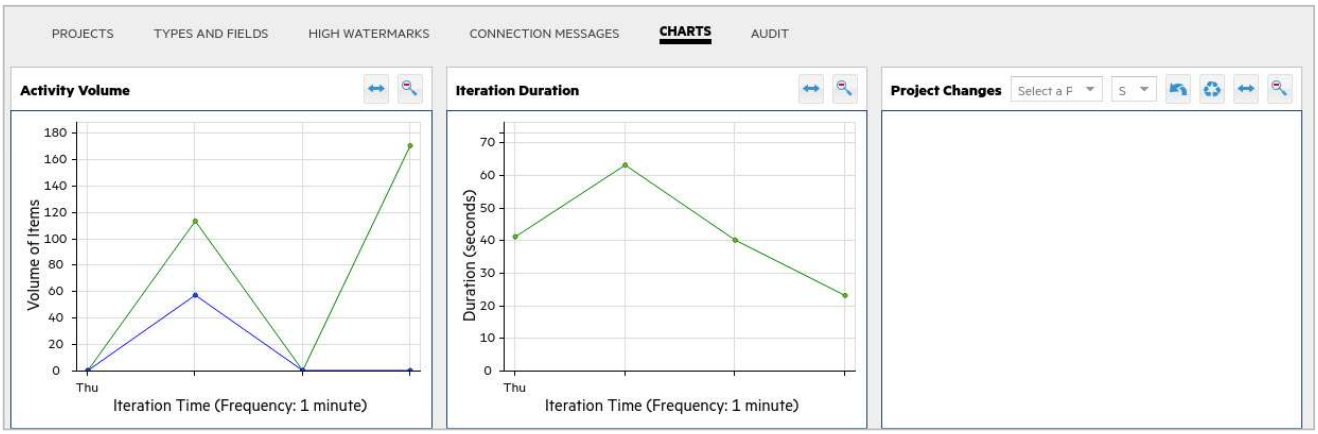
The **CONNECTION MESSAGES** tab shows a summary log of the synchronizations that have occurred. A list of the different types and the number of items found and synced is shown, along with any error messages.



CHARTS

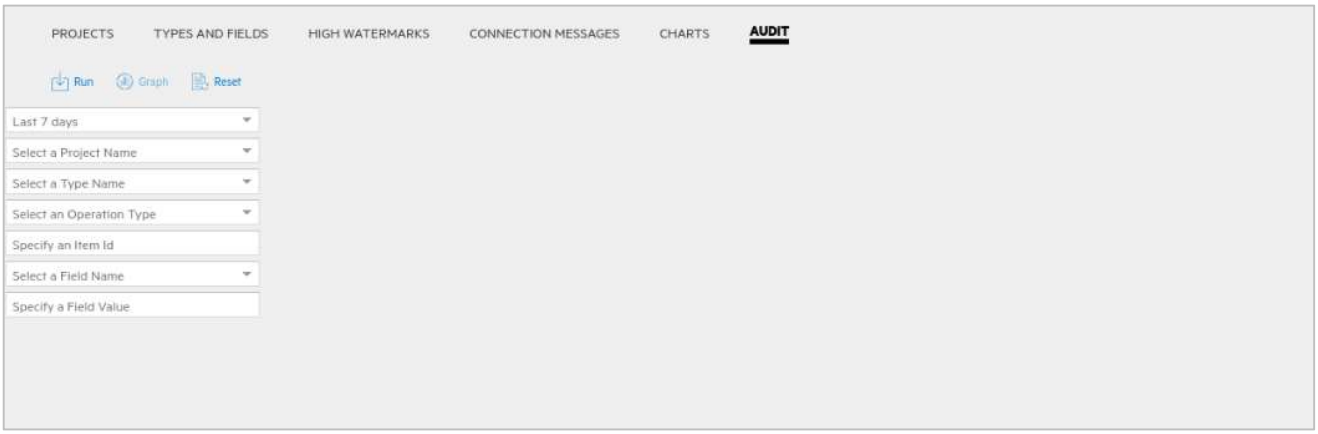
Charts track volume, changes, and errors. Each item is denoted by a specific colored line:

- Green: denotes volume, indicating the quantity of items read.
- Blue: denotes changes, indicating the quantity of items modified.
- Red: denotes errors, indicating failed changes.
- Hover over a data point to view detailed information.



AUDIT

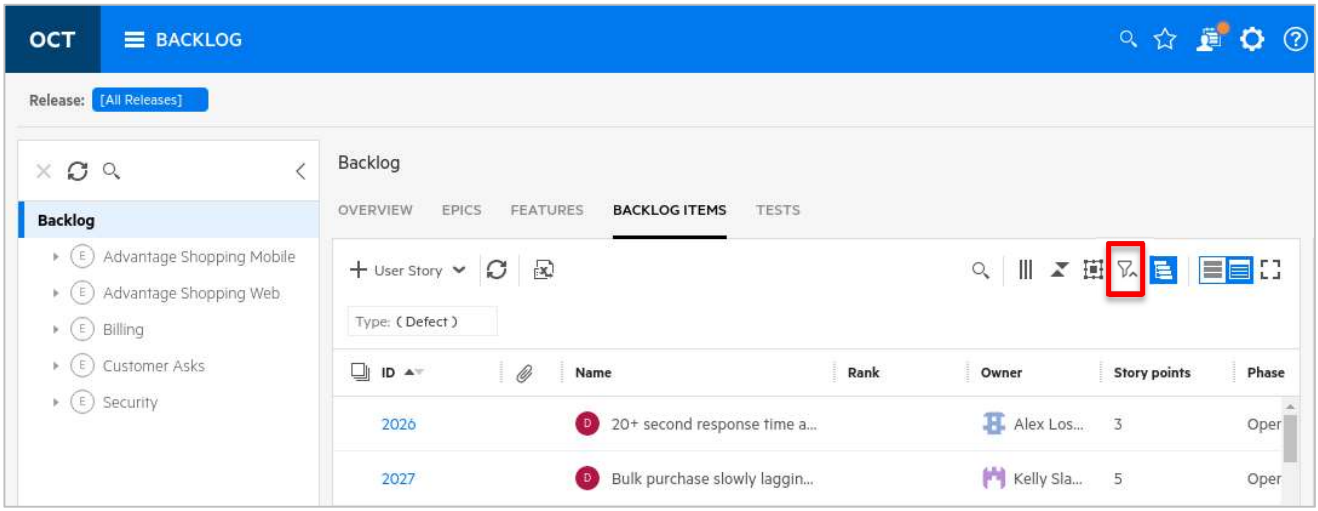
The **AUDIT** shows the details of what was added, updated or failed to sync during a given iteration for a given connection. In the case of sync failures, there will be a stack trace that you can click into and get full details.



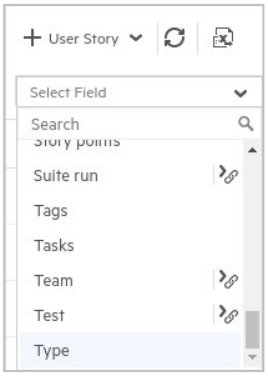
Verifying data in Octane and Jira

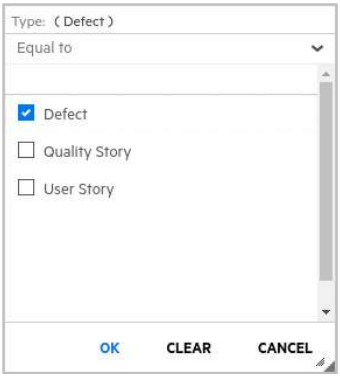
The next step is to verify that the data in Octane has been successfully synced to Jira.

1. In ALM Octane, navigate to the **Backlog** module and select the **BACKLOG ITEMS** tab.



2. Set a filter and select **Type** and filter for **Defects**.

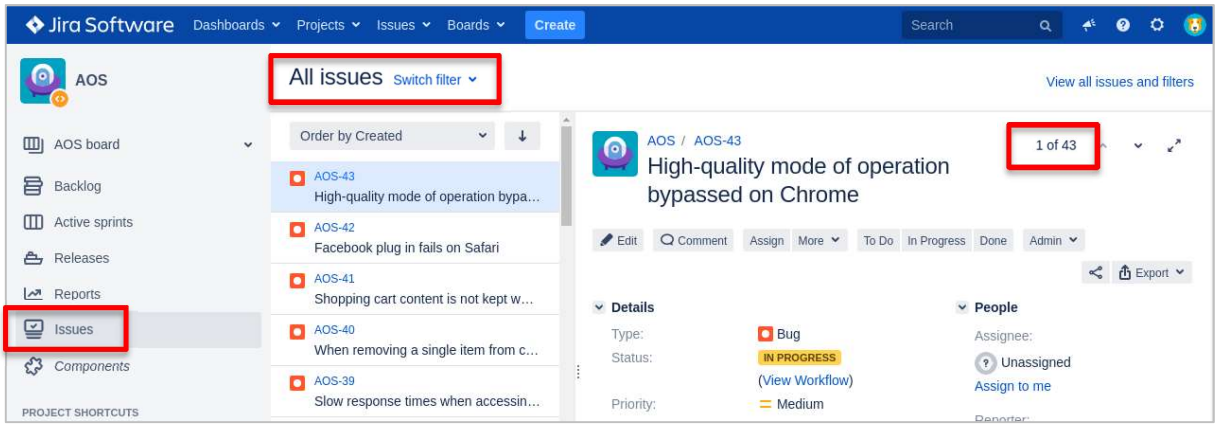




3. Look at the bottom of the grid view and note the total number of backlog items that are defects.

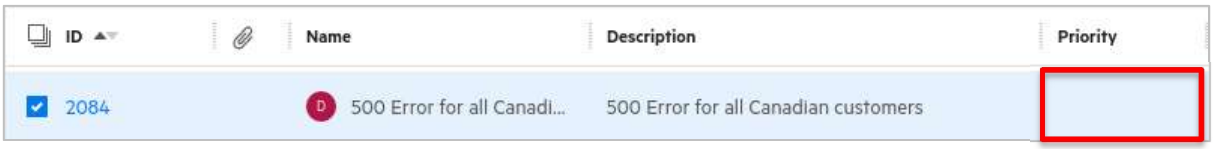
4. Open Jira and select the **AOS** project.

5. Select the **Issues** icon on the left edge and switch the filter to show **All Issues**.

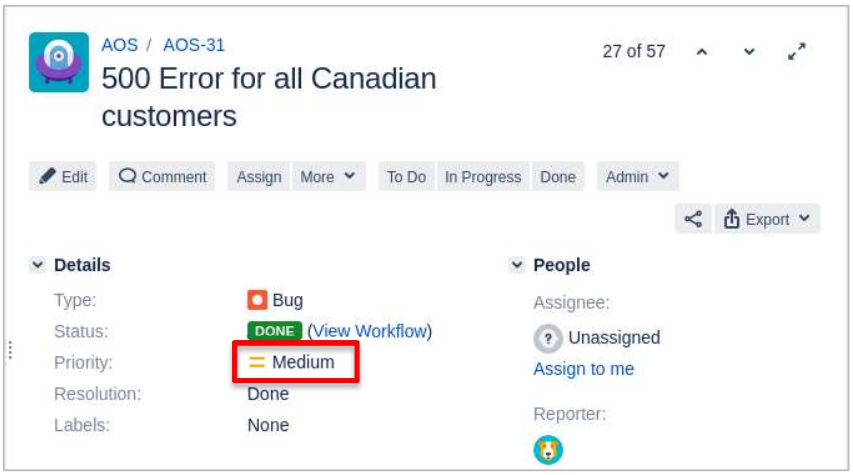


6. Verify that the number of issues in Jira is the same as the number of defects in Octane.

7. In **Octane**, scroll through the defects until you find one labeled: **500 error for all Canadian**…
   Note that in Octane there is no **Priority** set.

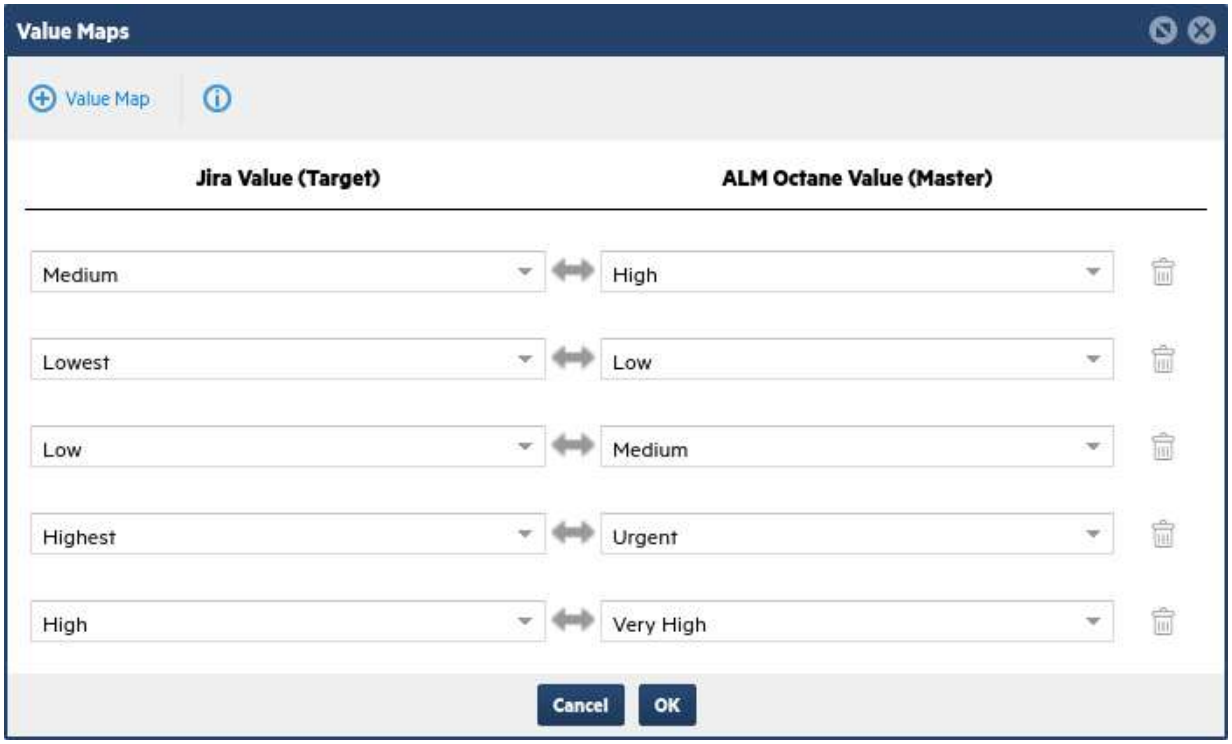


8. Switch to **Jira** and locate the same issue and note the **Priority** shown. Why is it different?



The priority is different because Jira requires a priority to be set and the default priority is **Medium**.

9. The table below shows the exact mappings of **Priority** field values from Jira to Octane. You can see or change these values if you choose to edit a connection.



Challenge Exercise #1

Change the Nimbus Octane Jira connection to allow bidirectional synchronization of Stories and Statuses. Verify your results by creating stories and defects in Octane and synchronizing them.

Challenge Exercise #2

Create a new connection and synchronize Epics and Features.

Synchronizing Jira and ALM QC

Now that we have some data in Jira let's also sync that to ALM QC.

1. Verify that you have both **NimbusServer** and **NimbusClient** up and running.

2. Start or verify the following Docker containers for this exercise.
   (The AutoPass container also needs to be running, but it automatically starts with Nimbus Server.)
   ```
   $ nimbusapp alm:15.0.1 start
   $ nimbusapp mfconnect:4.2 start
   $ nimbusapp jira:8.1.0 start
   ```

3. **Data Source Configuration**: In ALM, you must have a user with the default username "admin" and the default password of "Password1". Additionally, you must have the preconfigured domain and projects that come as part of the ALM container. These have been set up for you.

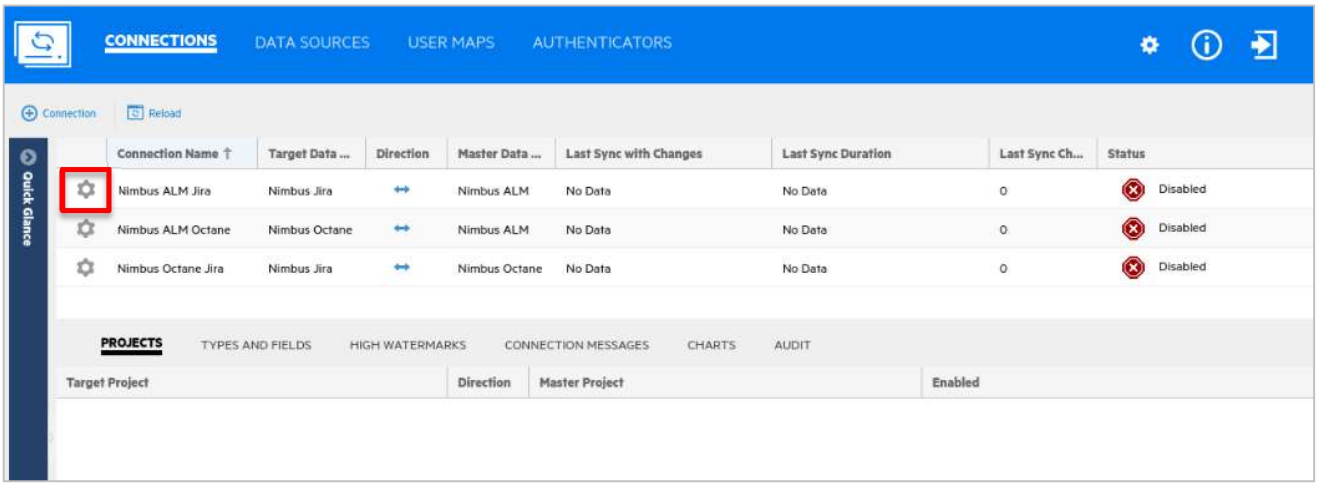
4. Switch over to the **NimbusClient** VM and open Internet Explorer and log in to **ALM Desktop Client**:
   Username: **admin**
   Password: **Password1**
   Domain: **DEFAULT**
   Project: **AOS**

5. Select the **Defects** module and click on **Select Columns**  to set the following columns order:
   1. **Summary**
   2. **Priority**
   3. **Status**
   4. **Severity**

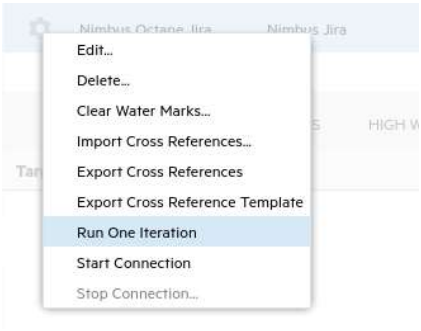
   This will make it easier to verify the defects when they come over from Jira.

6. Switch back to the **NimbusServer** VM and the **MF Connect** tab in your browser.

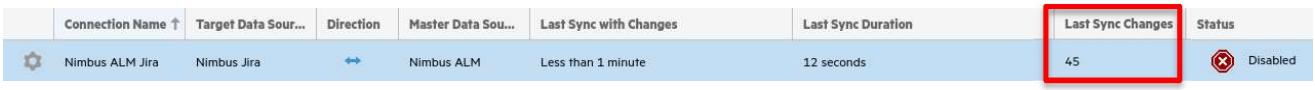
7. Select the **CONNECTIONS** tab at the top in **Connect**. You should see the three pre-configured connections in Connect:
   1. Nimbus ALM Jira – This syncs an ALM QC instance with Jira
   2. Nimbus ALM Octane – This syncs an ALM QC instance with Octane
   3. Nimbus Octane Jira – This syncs an Octane instance with Jira



8. **Left click** on the gear icon to the left of the **Nimbus ALM Jira** connection. Select **Run One Iteration**. This will run just one synchronization cycle. If you want it to run a continuous sync you would choose **Start Connection**.



6. The **Disabled** icon will turn green to show the synchronization cycle has started. It should take 30-90 seconds to finish the sync cycle.

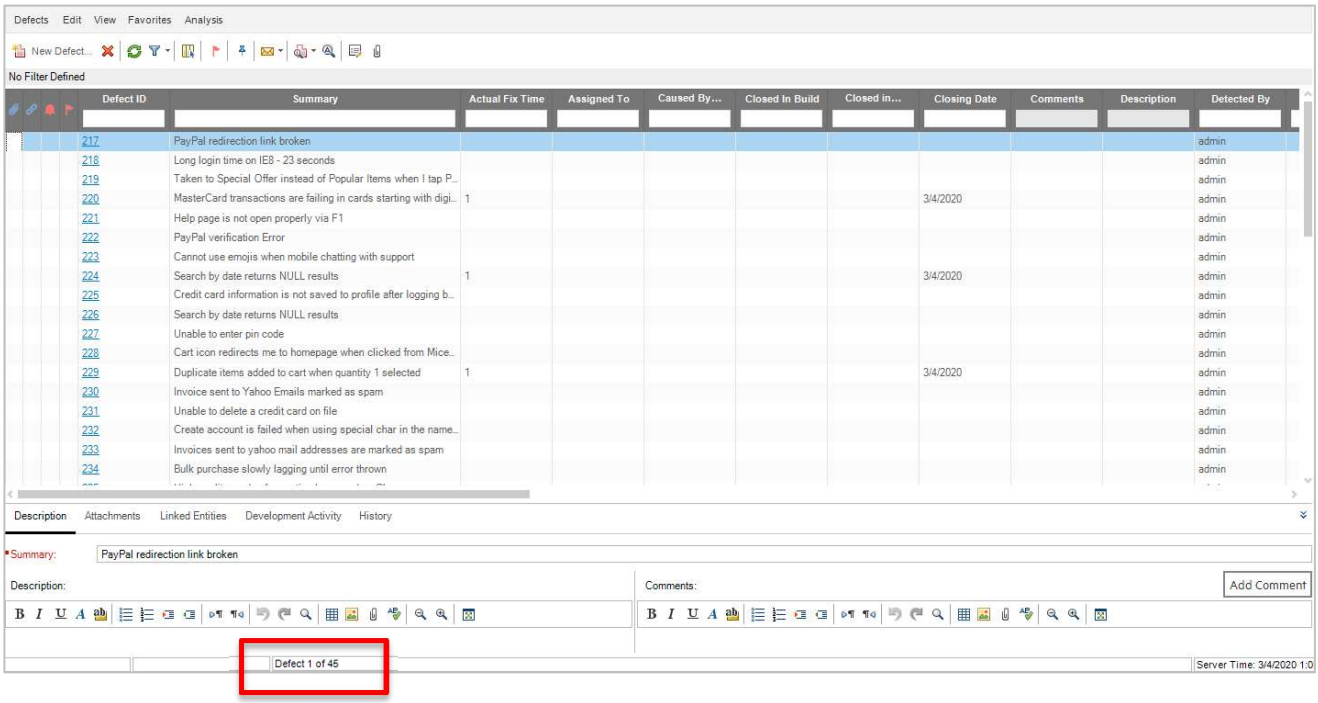


You can see how many items were synced under the **Last Sync Changes** column.

7. After a short time, click the **Reload** button to update your display. Wait until the synchronization has finished.

8. Switch back to **NimbusClient** and **IE** and click on the **Refresh** icon to see the defects. NOTE: the AOS project had no defects prior to this sync so these will all be new.

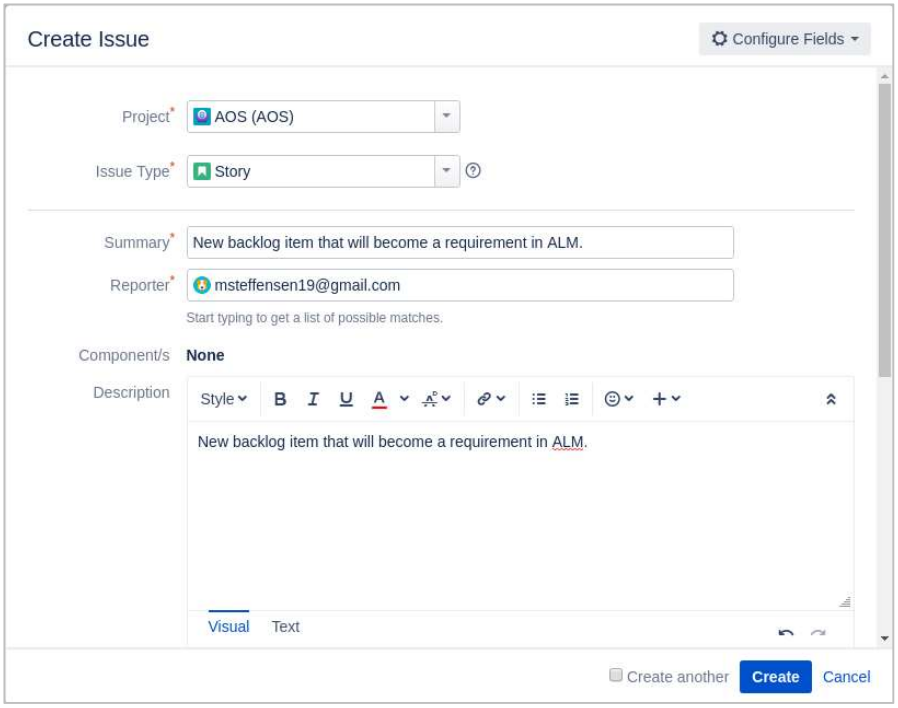
9. **Select** one of the defect rows.
Note the number of defects that were sync'd by looking at the bottom of the table.
You should not have seen any stories come over from Jira since none exist.



10. Switch back to **NimbusServer** and **Jira** in your browser and click on the **Backlog** icon on the left.

11. Click **Create** to create a new **Backlog** item.

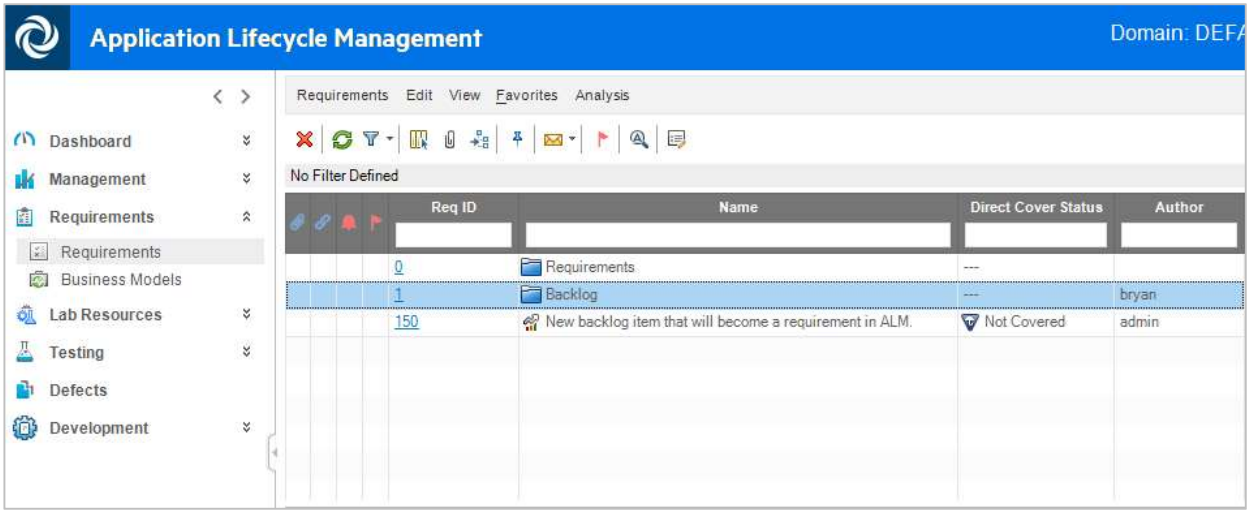
12. Create a new story item called: "**New backlog item that will become a requirement in ALM.**"



13. Go back to the **MF Connect** tab and select the **Nimbus ALM Jira** gear icon and **Run One Iteration**.

14. After the synchronization has finished, go back to **NimbusClient** and ALM and select the **Requirements** module.

15. Click the **Refresh** icon and verify that your new Jira story has been sync'd as a new requirement.



Challenge Exercise #3

Change the **Priority** of the new ALM requirement to **Low** and sync again. Verify that the corresponding story in Jira has its priority changed to **Lowest**.

Challenge Exercise #4

Create a Custom UDF in ALM called **Jira ID** on Defects and map Jira's ID to this field. Synchronize and verify that your new field is showing and set properly.

Synchronizing ALM QC and Octane

Next we'll look at synchronizing defects and requirements in ALM QC with ALM Octane.

1. Verify that you have both **NimbusServer** and **NimbusClient** up and running.

2. Start or verify the following Docker containers for this exercise.
   (The AutoPass container also needs to be running, but it automatically starts with Nimbus Server.)
   ```
   $ nimbusapp alm:15.0.1 start
   $ nimbusapp mfconnect:4.2 start
   $ nimbusapp octane:15.0.46.68 start
   ```

3. **NOTE**: if you previously sync'd defects into ALM, reset the container state with the following commands:
   ```
   $ nimbusapp alm:15.0.1 down
   ```
   (Choose Yes if asked to remove the container)
   ```
   $ nimbusapp alm:15.0.1 up
   ```

4. **Data Source Configuration**: In ALM QC, you must have a user with the default username "admin" and the default password of "Password1". Additionally, you must have the preconfigured domain and projects that come as part of the ALM container. These are preconfigure in the standard ALM container.

   In ALM Octane, you must have a user with the default username "sa@nga" and the default password of "Password1". These are preconfigured in the standard Octane container.

   NOTE: Requirements sync is a BETA feature in the 4.2 release of MF Connect

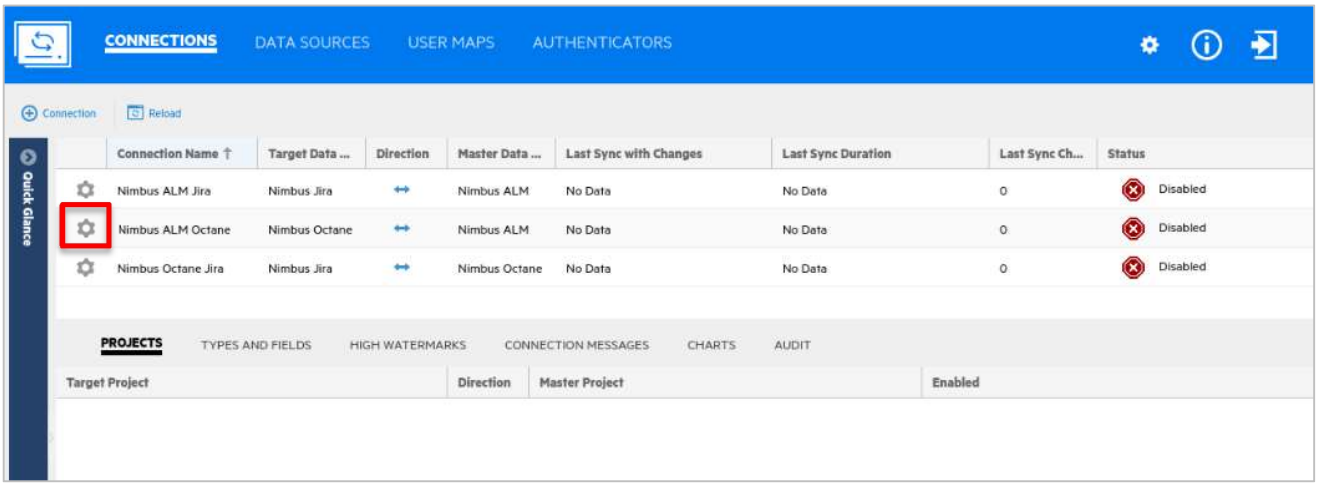
5. Switch over to the **NimbusClient** VM and open Internet Explorer and log in to **ALM Desktop Client**:
   Username: **admin**
   Password: **Password1**
   Domain: **DEFAULT**
   Project: **AOS**

6. Select the **Defects** module and click on **Select Columns**  to set the following columns order:
   1. **Summary**
   2. **Priority**
   3. **Status**
   4. **Severity**

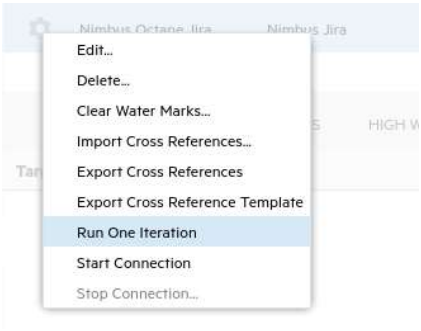
   This will make it easier to verify the defects when they come over from Octane.

7. Switch back to the **NimbusServer VM** and the **MF Connect** tab in your browser.

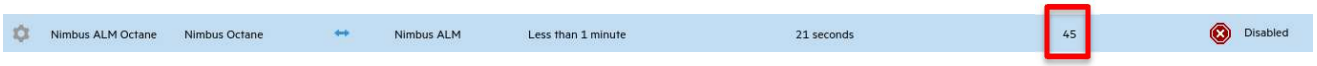
8. Select the **CONNECTIONS** tab at the top in **Connect**. You should see the three pre-configured connections in Connect:
   1. Nimbus ALM Jira – This syncs an ALM QC instance with Jira
   2. Nimbus ALM Octane – This syncs an ALM QC instance with Octane
   3. Nimbus Octane Jira – This syncs an Octane instance with Jira



9. **Left click** on the gear icon to the left of the **Nimbus ALM Octane** connection. Select **Run One Iteration**. This will run just one synchronization cycle. If you want it to run a continuous sync you would choose **Start Connection**.



9. The **Disabled** icon will turn green to show the synchronization cycle has started. It should take 30-90 seconds to finish the sync cycle.

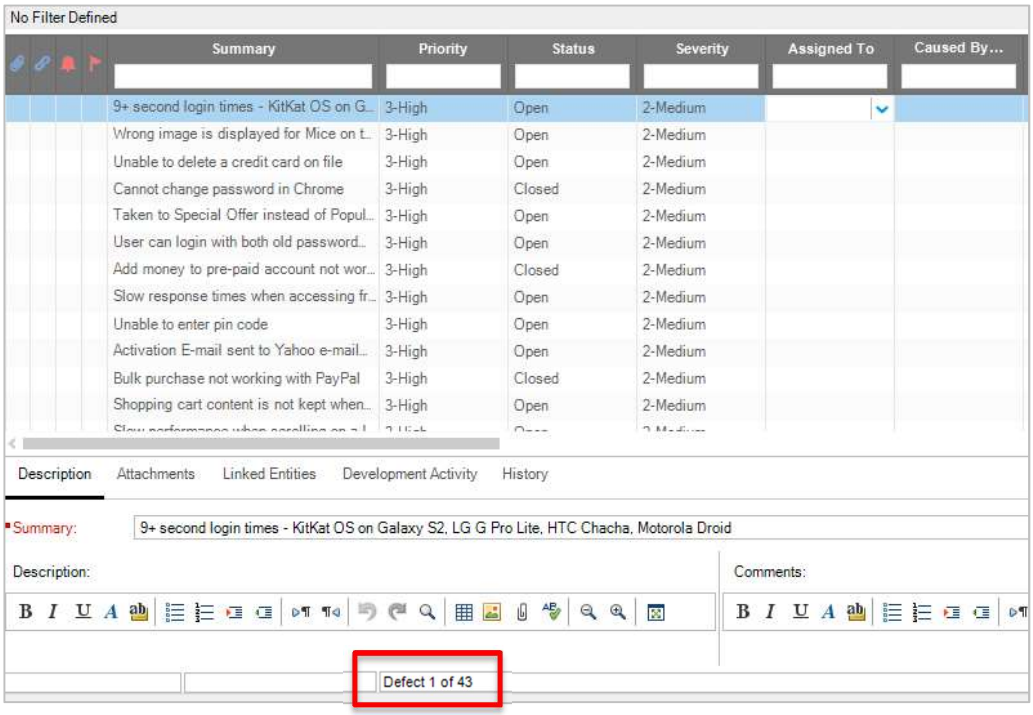


You can see how many items were synced under the **Last Sync Changes** column.

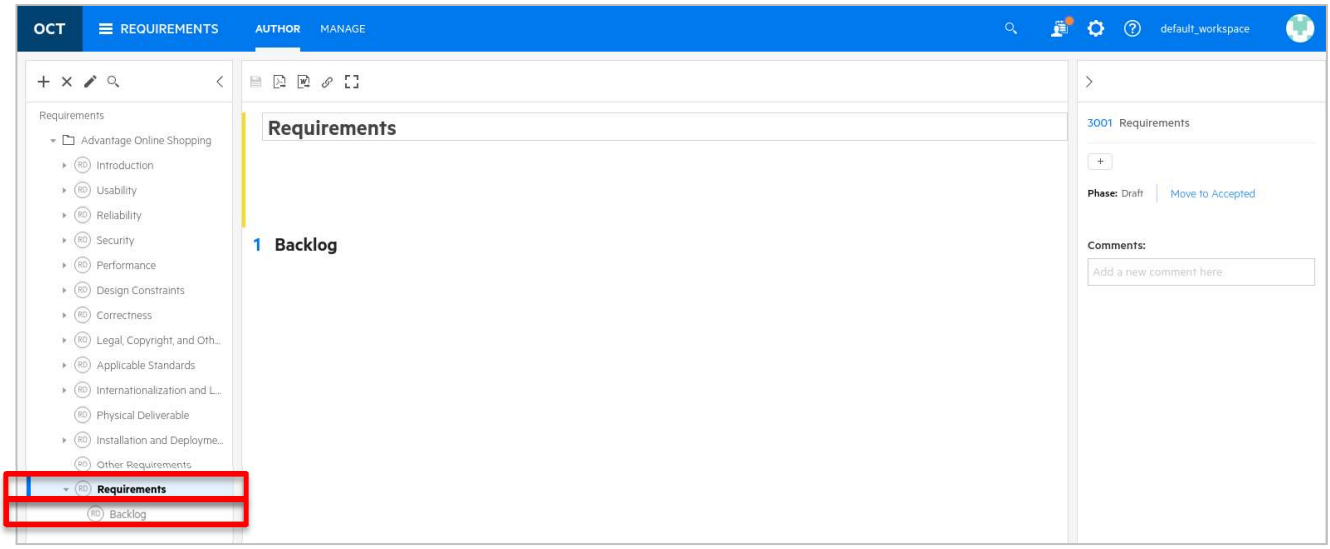
10. After a short time, click the **Reload** button to update your display. Wait until the synchronization has finished.

11. Switch back to **NimbusClient** and **IE** and click on the **Refresh** icon to see the defects.

12. **Select** one of the defect rows.
Note the number of defects that were sync'd by looking at the bottom of the table.
You should not have seen any stories come over from Jira since none exist.



13. Switch back to **NimbusServer** and **Octane** in your browser and click on the **Requirements** module. At the bottom of the page you should see a new category and sub-requirement as shown.



14. The number of items sync'd should be two more than the number of defects since these two requirements were brought from ALM into Octane. This connection moves requirements from ALM to Octane and sync's bi-directionally the defects in each.

Exercise 17: Integrating SonarQube Code Coverage

ROLE: INTEGRATION OR AUTOMATION ENGINEER

Code Coverage provides an automated and impartial way to judge the quality of code. Tools like SonarQube and JaCoCo are often used to check the quality of code and determine if the embedded unit tests actually cover enough of the code. They also provide a variety of other code quality metrics.

Jenkins offers integration with these tools and ALM Octane can integrate this data into charts that show a summary of coverage by pipeline run and by package

In this exercise we'll look how to configure this integration.

Running a SonarQube Code Scan

1. We need to have the following VM's running for this exercise:
   - NimbusServer

2. Run the following commands or verify that these containers are already running.
   Allow enough time for these containers to settle down (5-8 minutes or so).

```
$ docker login        (Your instructor will provide the credentials)
$ nimbusapp aos:2.2 start
$ nimbusapp devops:2.2.1 start
$ nimbusapp intellij:2.2.0 start
$ nimbusapp sonarqube:7.7 start
$ nimbusapp octane:15.0.46.68 start
```

3. From **NimbusServer**, start Chrome and open the **Jenkins** shortcut and open the **ALL** tab.

4. Make a copy of the **AOS_Web_Root_Module_Pipeline** job called **AOS_Web_Root_Module_SonarQube_Pipeline.**

5. Edit the Jenkins configuration for **AOS_Web_Root_Module_SonarQube_Pipeline** and replace the **SonarQube Analysis For ALM Octane** stage with the following code:

```
stage('SonarQube Analysis For ALM Octane')
{
  agent { label 'master'}
  when {
        expression {
                    return  sh (
                    script: "curl -L -s --head  --request GET  http://nimbusserver.aos.com:8020 |
grep '200'",
                    returnStatus: true
                              ) == 0
                  }
      }
  steps {
         withSonarQubeEnv ( "SonarQube" ) {
```

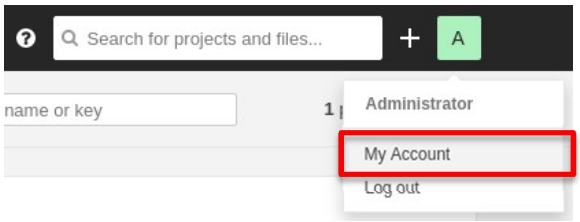
```
            addALMOctaneSonarQubeListener pushCoverage: true, pushVulnerabilities: true,
sonarServerUrl:env.SONAR_HOST_URL, sonarToken:env.SONAR_AUTH_TOKEN
            copyArtifacts filter: '**/jacoco.xml', fingerprintArtifacts: true, flatten: false,
projectName: 'aos-web-build-root', selector: lastSuccessful()
            sh ( script: "ls -la" )
            sh "mvn sonar:sonar -P DA -pl root -am -Dsonar.login=$SONAR_AUTH_TOKEN \
               -
Dsonar.coverage.jacoco.xmlReportPaths=${WORKSPACE}/common/target/site/jacoco/jacoco.xml,${WORKSPA
CE}/root/target/site/jacoco/jacoco.xml \
                  -Dsonar.host.url=http://nimbusserver.aos.com:8020 -
Dsonar.analysis.buildNumber=${currentBuild.number} \
                  -Dsonar.analysis.jobName=${JOB_NAME} -Dsonar.test.inclusions=**/*Tests.java -
Dsonar.java.binaries=/ \
                  -
Dsonar.exclusions=root/src/main/webapp/scripts/nv_files/**,root/src/main/webapp/languages/english
.js,root/src/main/webapp/Gruntfile.js,root/src/main/webapp/app/**/index.js,**/module.js,**/main.j
s,root/src/main/webapp/utils/noUiSlider.8.2.1/nouislider.js,root/src/main/webapp/r.js,root/src/ma
in/webapp/api/**"
                                                  }
        }
}
```

6. Click **Save** to save your configuration changes.

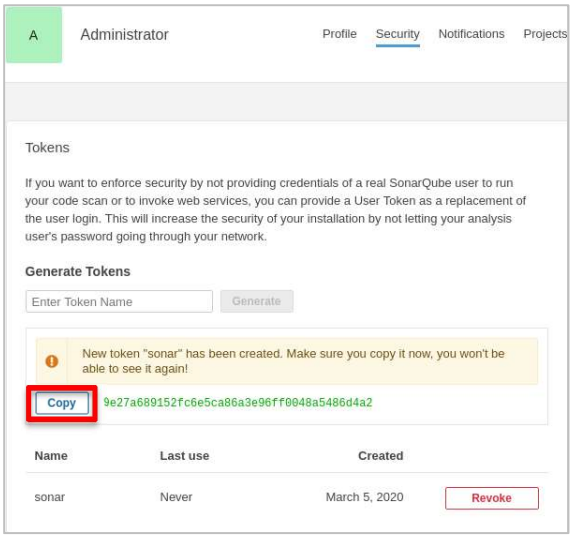
7. In Chrome, open the **SonarQube** URL (http://nimbusserver.aos.com:8020)

8. In the upper right corner, log in as Administrator with
   Login: admin
   Password: admin

9. Select **My Account** under the **A** pull-down.

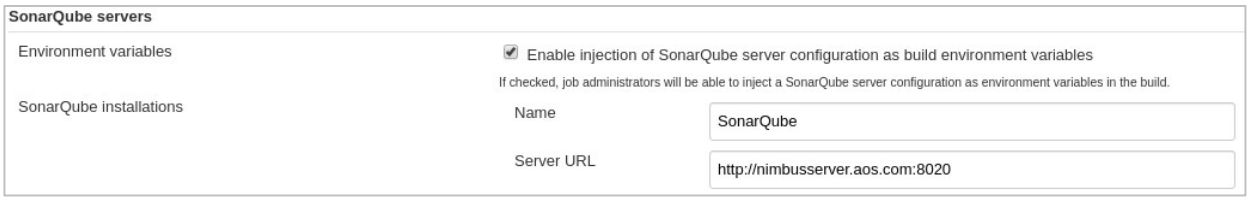


10. Select the **Security** tab and **generate** a new token called **sonar.**
    Copy the token that is generated.



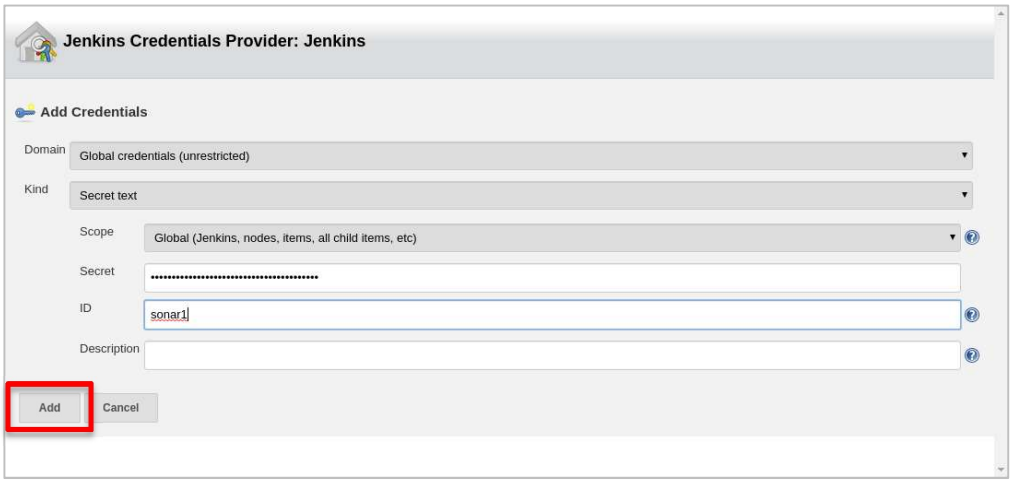
11. Switch back to Jenkins and select the **Manage Jenkins** and **Configure System**.
12. Locate the section for **SonarQube servers**.
13. Set the following:

    - Enable injection checkbox is **checked**
    - Name: **SonarQube**
    - Server URL: http://nimbusserver.aos.com:8020



14. Under the **Add** pull-down select Jenkins.

15. Select **Kind** to be **Secret Text** and set the following fields:
    Secret: **<paste in the value from step 10 above>**
    ID: **sonar1**



16. Click **Add** to create this credential.

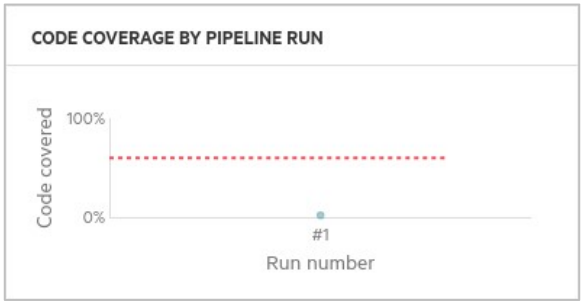
17. From the pull-down select **sonar1** and click **Save**.

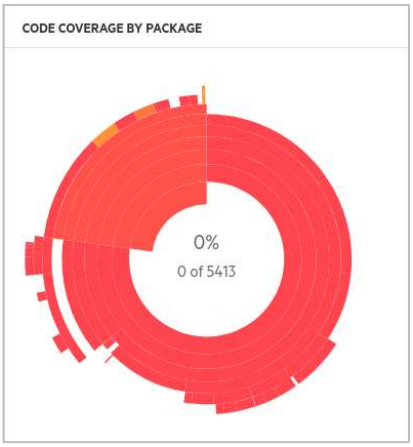
18. Open the Octane shortcut and create a new Pipeline in Octane from **AOS_Web_Root_Module_SonarQube_Pipeline**

19. Run the **AOS_Web_Root_Module_SonarQube_Pipeline** pipeline from Octane or Jenkins.

20. When the pipeline finishes, add the following two new charts to this pipeline:
    - Code coverage by Pipeline Run
    - Code Coverage by Package

21. You should see charts similar to the following:





22. To see the full SonarQube results, click the SonarQube icon from the Jenkins **Build History** page.

Exercise 18: Building the Account Service Module

In our current **AOS_Web_Root_Module_Pipeline**, we build and deploy the root module. That's just one of eight modules contained in the AOS application. In this section we look at how we can change this process to build and deploy the accountservices module instead. Of course we could even build and deploy all eight modules but during a demo this might take too long so we've condensed this process to just focus on one module at a time.

By building a new accountservice module, we can verify changes in the accountservice module when it is virtualized with Service Virtualization. Once the accountservice service is virtualized, we can effectively turn off this container and run with just the virtualized service.

Cloning the Root Module Jobs in Jenkins

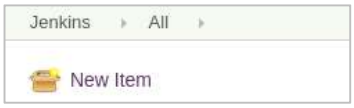
In the section we'll make copies of a Jenkins **job** and **pipeline** and modify them to work with accountservice.

1. From the NimbusServer VM, start (or verify) the **AOS**, **DevOps** and **IntelliJ** set of containers by typing:

   ```
   $ nimbusapp aos:2.2 start
   $ nimbusapp devops:2.2.1 start
   $ nimbusapp intellij:2.2.0 start
   ```

2. From the NimbusServer VM, open a browser to the **Jenkins** home page and select the **All** tab and locate the job called **AOS_Web_Build_Root** (don't click on it).
   We're going to make a **clone** of this job to build the accountservice module instead of Root.

3. Select the **New Item** option on the left alley.

4. At the top input box, name the new item **AOS_Web_Build_Account**
   We'll be reusing this name in another location so it's important to use this exact name.

5. Scroll to the bottom and in the **Copy from** field enter **aos-web-build-root**

6. Click **OK**.

7. You'll now be configuring your new **AOS_Web_Build_Account** Jenkins job.
   Scroll down to the Build section and enter the following for the maven command:

   ```
   clean install -P DA -pl accountservice -am
   ```

We're basically replacing the compilation of the root module with accountservice.

Thanks to **Maven** (which maps out all the module dependencies) we can easily change the module we want to build. Maven uses a file called pom.xml to contain those dependencies.

The output of this build is the accountservices.war file and this gets deployed on a Tomcat server.

8. Click **Save** to save your changes.
9. Navigate back to the **Jenkins home page** after saving your changes.
10. Select the **New Item** option again on the left alley. This time we'll clone a pipeline.
11. At the top, name the new item **AOS_Web_Account_Module_Pipeline**
12. Scroll to the bottom and in the **Copy from** field enter **aos-web-root-module-pipeline**
13. Click **OK**.

Updating the Pipeline Script

1. You'll now be configuring your new **AOS_Web_Build_Account** Jenkins pipeline. Scroll down to the **Pipeline Script** section and **replace** the code with the following:

```
def nodetest() {
    echo ('alive on $(hostname)')
}

pipeline{
    agent none
    environment {
        AOS_VERSION=get_aos_version()
                }
    stages{
        stage('Checkout')
                {
                    agent { label 'master'}
                    steps{
                        git branch: "${AOS_VERSION}", url: '/gitrepo/aos-source'
                    }
                }

        stage('Build Account Module')
                {
                    steps{
                        //=== AOS Web Build Account Module ===
                        //=== This build requires the AOS containers and the Inteliij container
===
                        build job: 'aos-web-build-root'
                        build job: 'AOS_Web_Build_Account'
                        }
                }
```

```
        stage('Deploy Account Module')
                {
                    agent { label 'master'}
                    steps{
                        script {
                            //undeploy accountservice module from Tomcat
                            httpRequest authentication: '94c1b1e6-5013-4d2d-a78a-a9b2b8adc9e5',
responseHandle: 'NONE', url: 'http://aos-accountservice:8080/manager/text/undeploy?path=/'
                            //copy artifacts from build
                            copyArtifacts filter: '**/*.war', fingerprintArtifacts: true,
flatten: true, projectName: 'AOS_Web_Build_Account/com.advantage.online.store:accountservice',
selector: lastSuccessful()

                            //undeploy root module from Tomcat
                            httpRequest authentication: '94c1b1e6-5013-4d2d-a78a-a9b2b8adc9e5',
responseHandle: 'NONE', url: 'http://aos-main:8080/manager/text/undeploy?path=/'
                            //copy artifacts from build
                            copyArtifacts filter: '**/*.war', fingerprintArtifacts: true,
flatten: true, projectName: 'aos-web-build-root/com.advantage.online.store:root', selector:
lastSuccessful()

                            //update configuration
                            configFileProvider([configFile(fileId: 'b9df64b1-0f01-4587-8f2a-
fda77d8f7cf3', targetLocation: './config-aos-war.sh')]) {
                            sh 'chmod +x ./config-aos-war.sh && ./config-aos-war.sh'
                              }
                        //deploy to Tomcat
                        sh "curl -v -u tomcatadmin:Password1 -T account*.war 'http://aos-
accountservice:8080/manager/text/deploy?path=/accountservice&update=true'"
                        sh "curl -v -u tomcatadmin:Password1 -T root*.war 'http://aos-
main:8080/manager/text/deploy?path=/ROOT&update=true'"
                        }

                }
        }
    }
}
def get_aos_version() {
    node('master') {
        return env.AOS_VERSION
    }
}
```

We're actually not changing much in this script – we're deleting the security and testing stages to simplify and speed things) and replacing the stage that gets built (**Deploy Account Module**) and also replacing the root module references with that of the accountservice module.

2. Click **Save** to save your changes.

Building the Account Services Module

1. You now have the pieces in place to build and deploy the accountservices service.

   Any changes you made to the AOS_Source in the accountservice module that is committed to the AOS source code will now get deployed as a modified version of the AOS website.

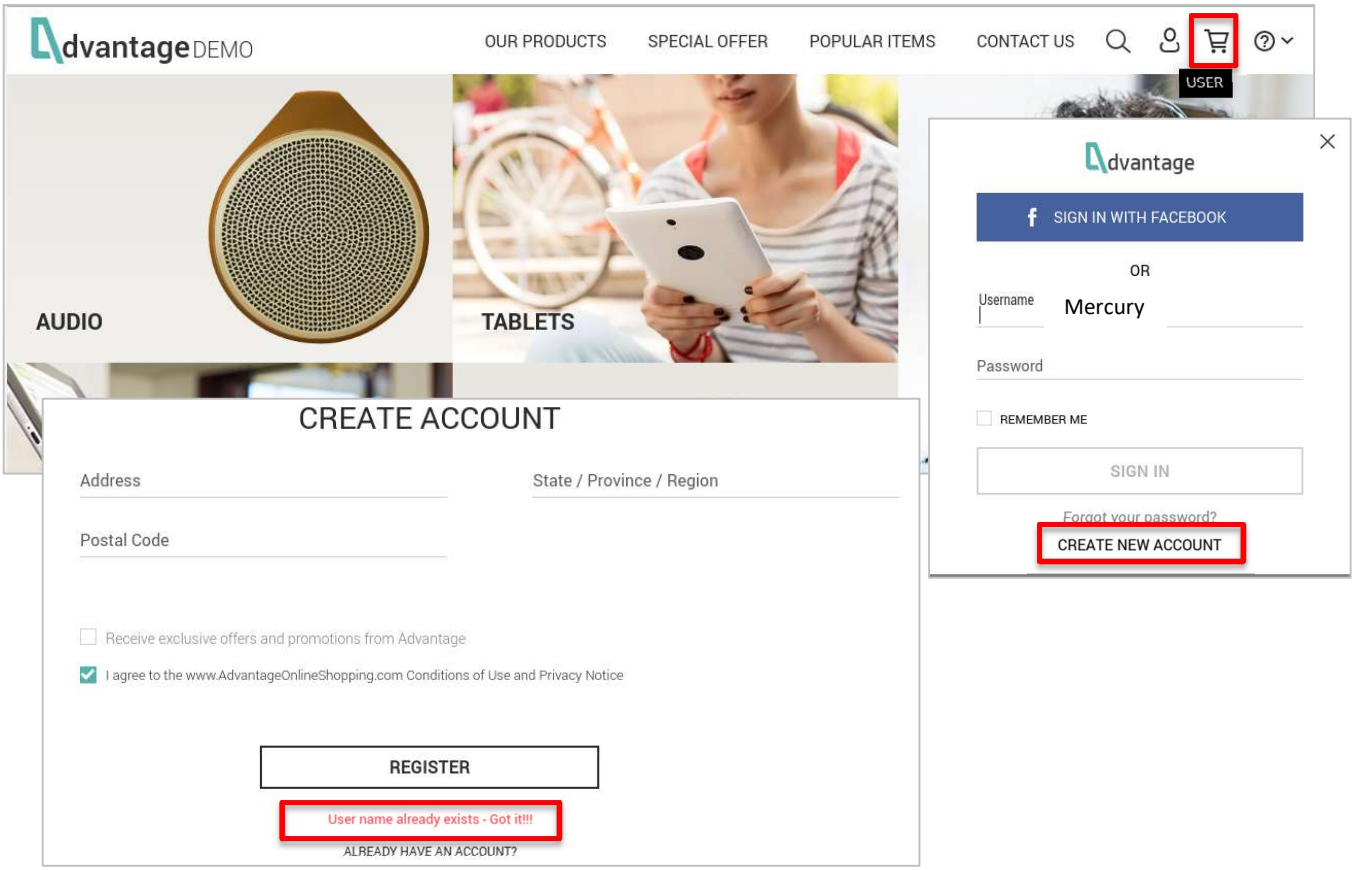
   For example, if you changed an error message and then triggered that message, the new error message would now appear.

2. In IntelliJ, open the AOS Source project and navigate to:
   aos_source/accountservice/src/main/java/com.advantage.accountsoap/model/Account.java

3. Change the error message from **line 54** in **Account.java** to something you would recognize like
   User name already exists – Got it!!!

   Don't forget to commit and push your changes.

4. From Jenkins, run the **AOS_Web_Account_Module_Pipeline** and then try creating a new account called **Mercury** (case sensitive). Since that account already exists you should see your new error message.



Appendix

Latest Container Versions	
Octane	15.0.46.68
DevOps	2.2.1
IntelliJ	2.2.0
AOS	2.2
Jira	8.1.0
MFConnect	4.2
SonarQube	7.7
SCA	19.1.2
SSC	19.1.0
Audit Workbench	19.1.2
ALM	15.0.1

CTRL+A + F9 to update entire document