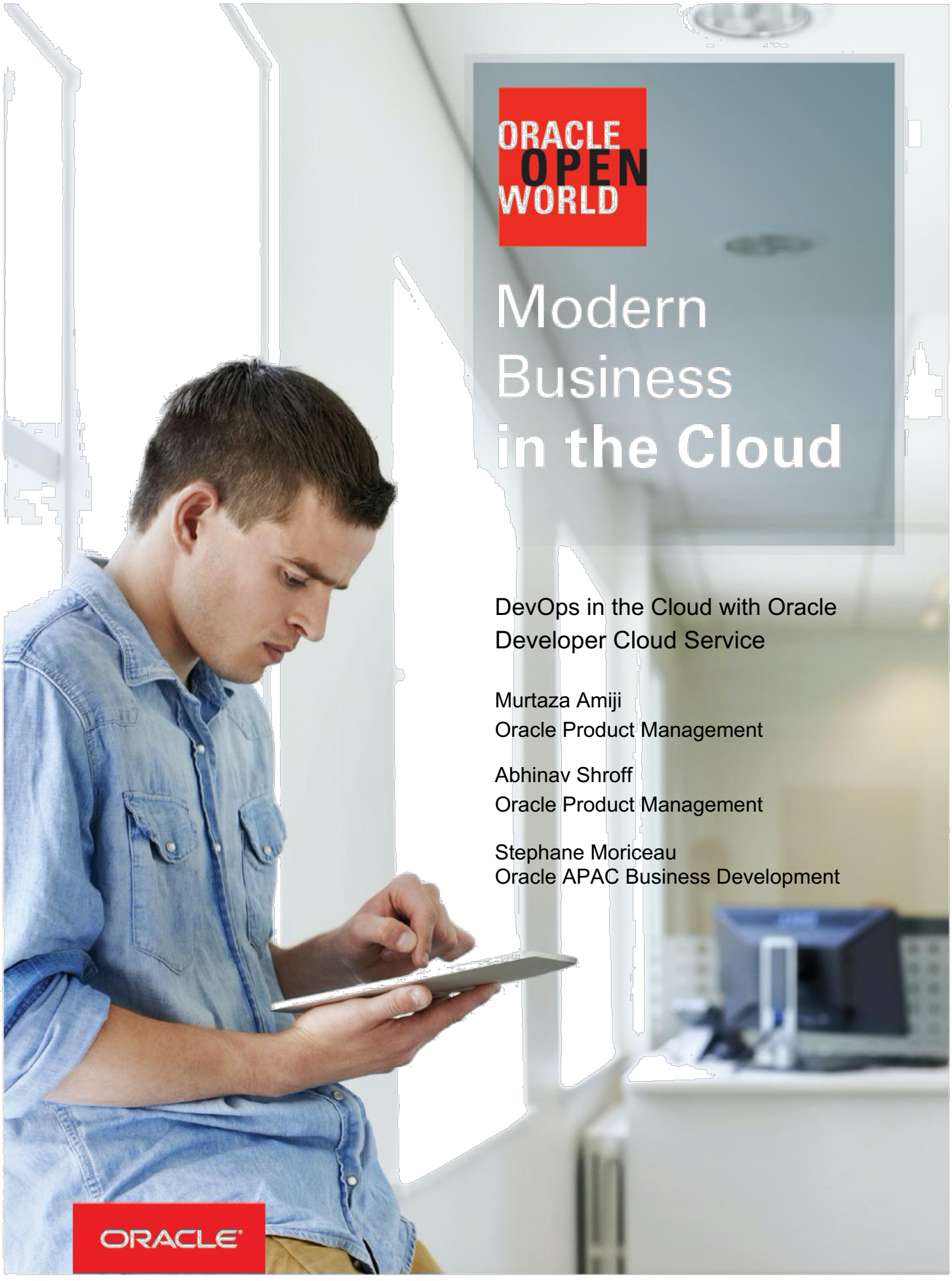

ORACLE
OPEN
WORLD
Modern
Business
in the Cloud
DevOps in the Cloud with Oracle
Developer Cloud Service
Murtaza Amiji
Oracle Product Management
Abhinav Shroff
Oracle Product Management
Stephane Moriceau
Oracle APAC Business Development
ORACLE

Oracle Cloud Infrastructure DevOps Automation

In this lab, you'll learn how to build a Docker image for a Node.js REST service on Oracle Developer Cloud Service (DevCS), and push it to Docker registries such as Oracle Cloud Infrastructure Registry and DockerHub.

Why Oracle Developer Cloud Service?

In recent years, the world of application development has adopted new methodologies that aim to improve the quality and speed with which applications are being delivered. The introduction of innovative development approaches, such as test-driven development and Agile development, gave rise to a set of new techniques and tools that enable those methodologies. Tools such as automatic build utilities combined with continuous integration platforms, as well as enhanced collaborative tools such as wikis and code review utilities, simplify the adoption of these new methodologies.

However, for many IT shops, setting up these environments and maintaining them was difficult and cost prohibitive, so many organizations stuck to the old way of building applications. Oracle has changed this by introducing a simpler way to adopt modern development methodology and tools with a cloud-based offering known as Oracle Developer Cloud Service.

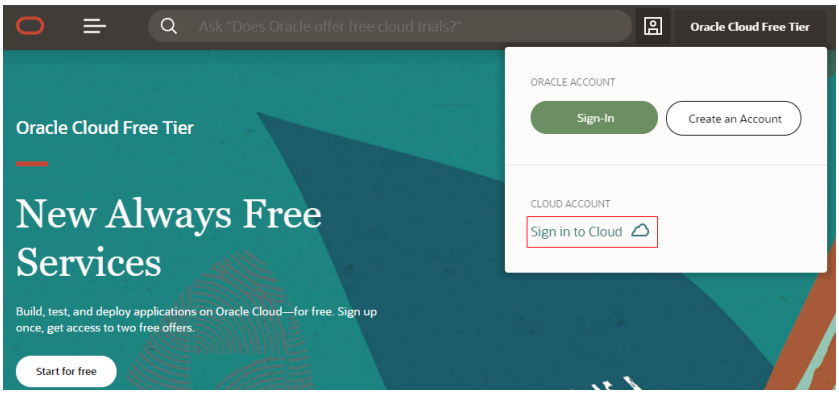
In this tutorial, you will work with DevCS and learn about some of its most important features. Specifically, you will:

- Create a DevCS *project*, which serves as the organizing principle for your software development efforts;
- Pull the code from an existing GitHub repository and use it to populate a DevCS-hosted Git repository;
- Create a task within DevCS and an Agile board to track it;
- Complete the task by branching, making changes to the code using the online editor, then merging the code;
- Build the Docker image with Continuous Integration in DevCS;
- Push the Docker image to Oracle Cloud Infrastructure Registry and DockerHub.

Getting Started –

Before we start with the lab, please follow the below instructions to setup you instance:

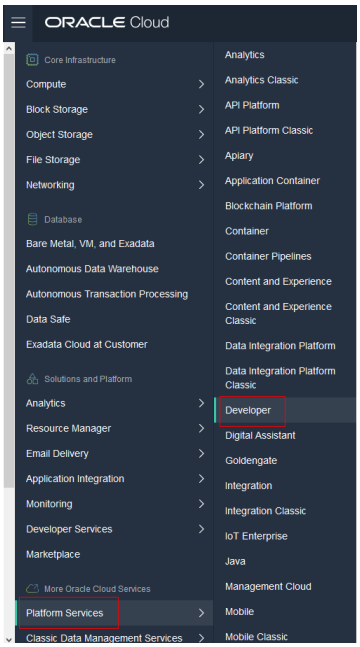
1. **Request a free account**



- Request a free Oracle Cloud account with credits from https://www.oracle.com/cloud/.\
- See the Overview of the Free Oracle Cloud Promotion infographic for more information.

2. **Create the DevCS service instance**

- When you get a *Welcome to Oracle Cloud* email, sign into your Oracle Cloud account using the details provided.
- On the Oracle Cloud page, open the navigation menu. Under **More Oracle Cloud Services**, select **Platform Services**, and then select **Developer**.



- Click **Create Instance**.
- On the Create New Instance page, enter the required details and click **Next**.
- On the Service Details page, verify the entered details and click **Next**.
- On the Confirmation page, click **Create**.

3. Set up Compute and Storage

The OCI Compute services provide the virtual machines (VMs) on which DevCS runs its builds. The OCI Storage services provide storage for build and Maven artifacts

- Follow this link to set up the compute service (Note: Ignore the "Set Up the OCI Classic Connection" section)

Now set up the storage service as follows:

- Open the navigation menu. Under **Core Infrastructure**, click **Object Storage**.
    - A list of the existing buckets in the compartment you're viewing is displayed.

- Select the compartment you created earlier from the **Compartment** list on the left side of the page.
    - A list of existing buckets is displayed.
- Click **Create Bucket**.
- In the **Create Bucket** dialog box, specify the attributes of the bucket:
    - **Bucket Name:** The system generates a default bucket name that reflects the current year, month, day, and time, for example **bucket-20190306-1359**.

      (Note: If you change this default to any other bucket name, use letters, numbers, dashes, underscores, and periods. Do not include any confidential information.)

    - **Storage Tier: Select the Standard option**.

      (Note: Available tiers include:

        - **Standard** is the primary, default Object Storage tier for storing frequently accessed data that requires fast and immediate access.
        - **Archive** is a special tier for storing infrequently accessed data that requires long retention periods. Access to data in the **Archive** tier is not immediate. You must restore archived data before it's accessible. For more information, see Overview of Archive Storage.)

    - **Object Events: Select the default option**.

      (Note: Select **Emit Object Events** if you want to enable the bucket to emit events for object state changes. For more information about events, see Overview of Events.)

- **Encryption:** Buckets are encrypted with keys managed by Oracle by default, but you can optionally encrypt the data in this bucket using your own Key Management encryption key. **Select the default option**.

  (Note: To use Key Management for your encryption needs, select **Encrypt Using Customer-Managed Keys**. Then, select the **Vault Compartment** and **Vault** that contain the master encryption key you want to use. Also select the **Master Encryption Key Compartment** and **Master Encryption Key**. For more information about encryption, see Overview of Key Management. For details on how to create a vault, see Managing Vaults.)

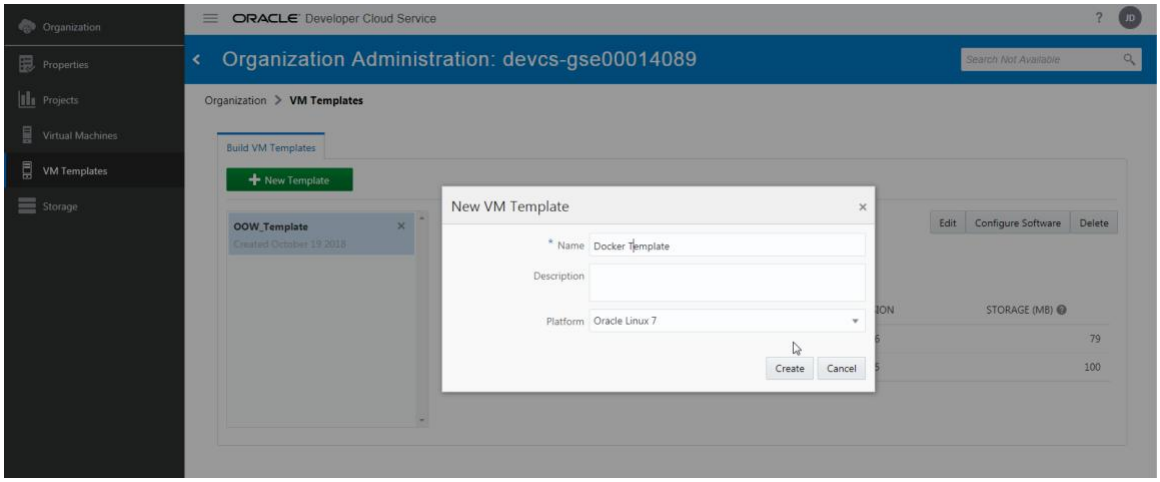
- **Tags:** Optionally, you can apply tags. If you have permissions to create a resource, you also have permissions to apply free-form tags to that resource. To apply a defined tag, you must have permissions to use the tag namespace. For more information about tagging, see Resource Tags. If you are not sure if you should apply tags, skip this option (you can apply tags later) or ask your administrator.

- Click **Create Bucket**.

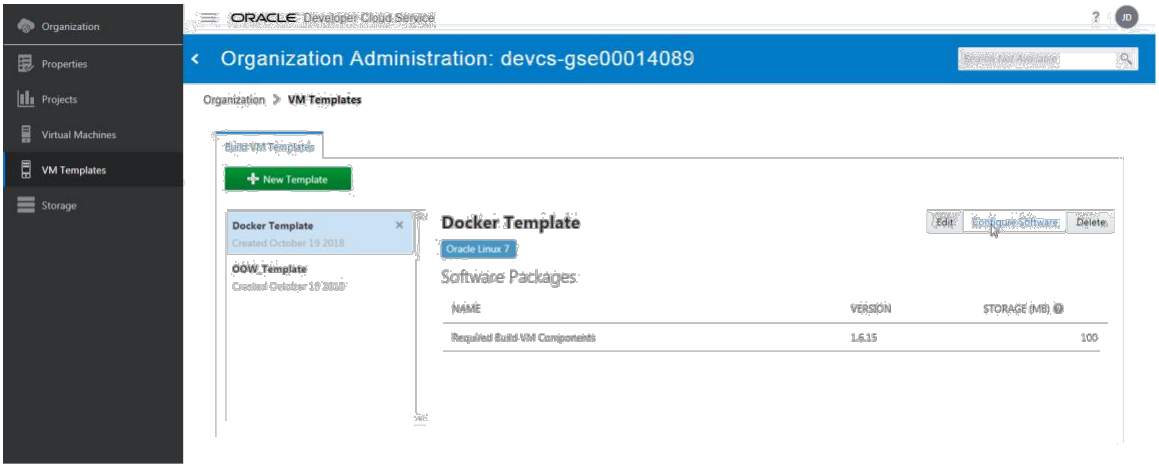
  - The bucket is created immediately and you can add objects to it. Objects added to archive buckets are immediately archived and must be restored before they are available for download.

4. For the builds to work in this lab, we will also have to configure Build Template and Build VM as per the below instructions:

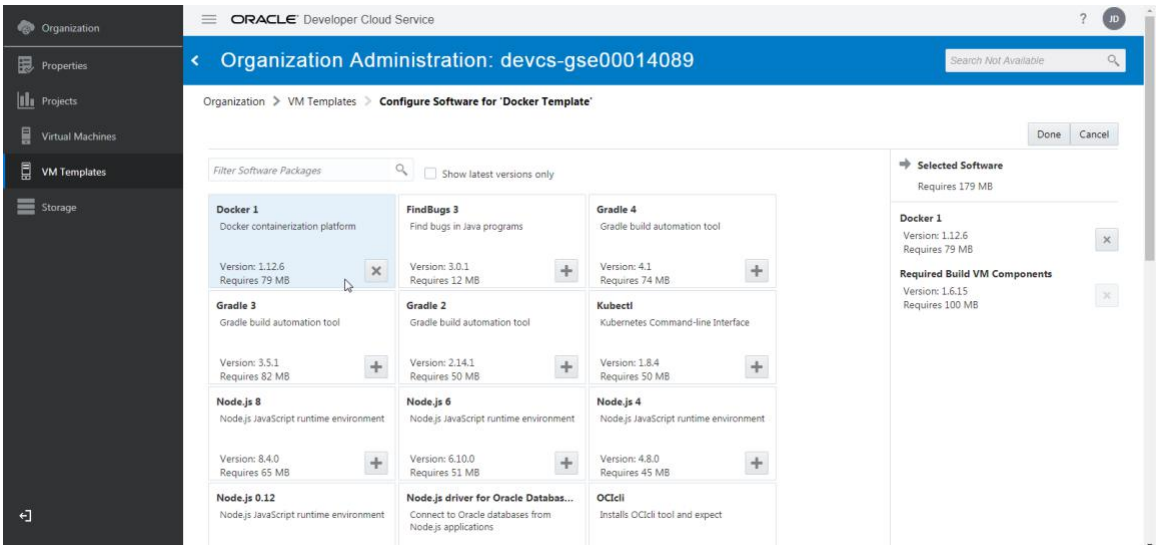
   1. In the Organization menu where you have configured the Storage and Compute Cloud Service Configurations. Click on the VM Templates tab and then on the “Create” button. In the dialog box that pops up, give a template name such as **DockerTemplate**. Select Platform as Oracle Linux 7 and click on the Create button.



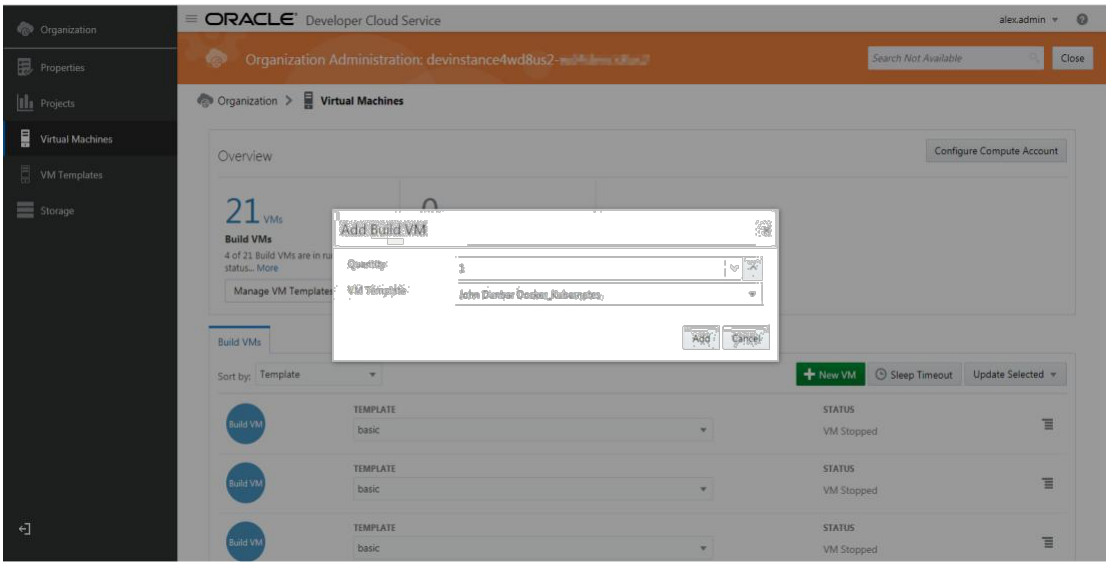
   2. Now on creation of the template click on “Configure Software” button.



3 Select Docker from the list of software bundles available for configuration and click on the + sign for each software bundle that you intend to add to the template. Then click on “Done” button to complete the Software configuration.



4 Click on the Build Virtual Machines tab, then click on “Create” button and in the dialog box that pops up, enter the Quantity as 1 and select Docker Template for the VM Template dropdown. Then click the Add button.



Now let's get started, with the lab!

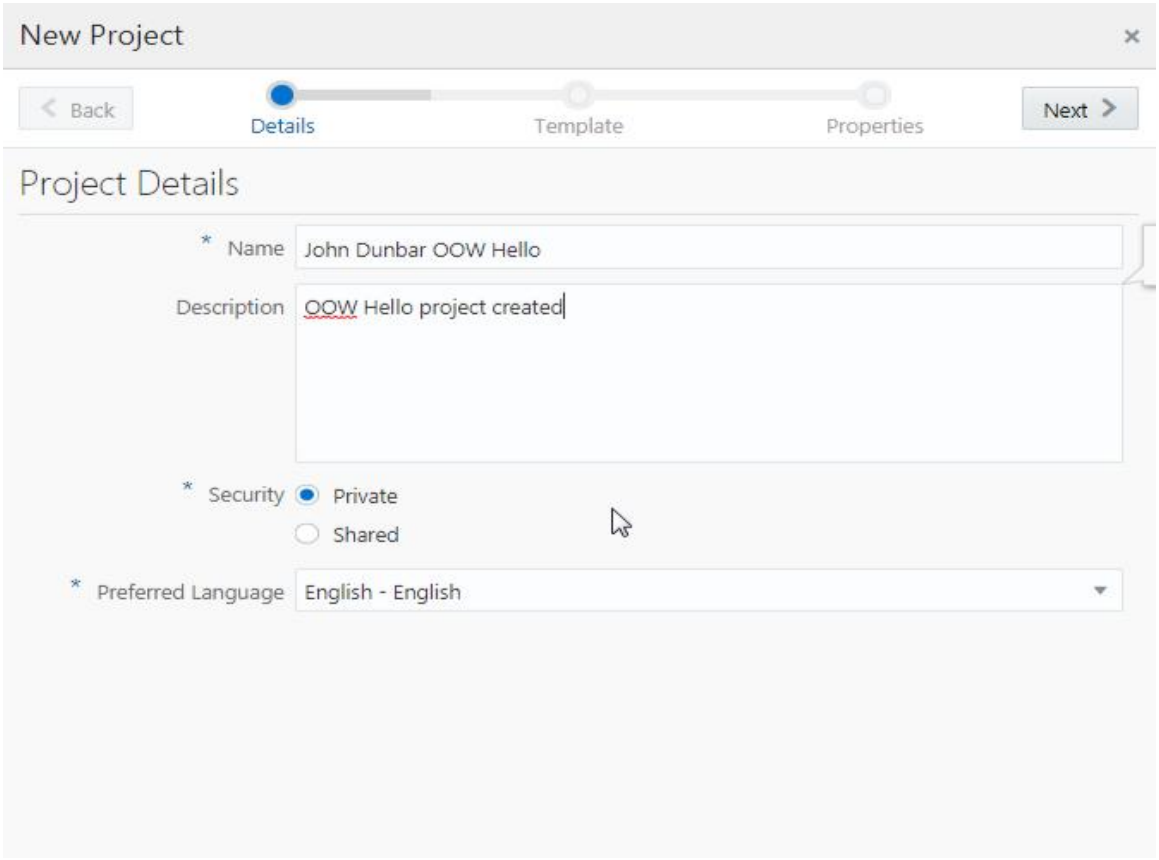
Milestone 1: Create a project environment for your team

In this section, you'll provision a complete development platform for your team by leveraging DevCS's web interface.

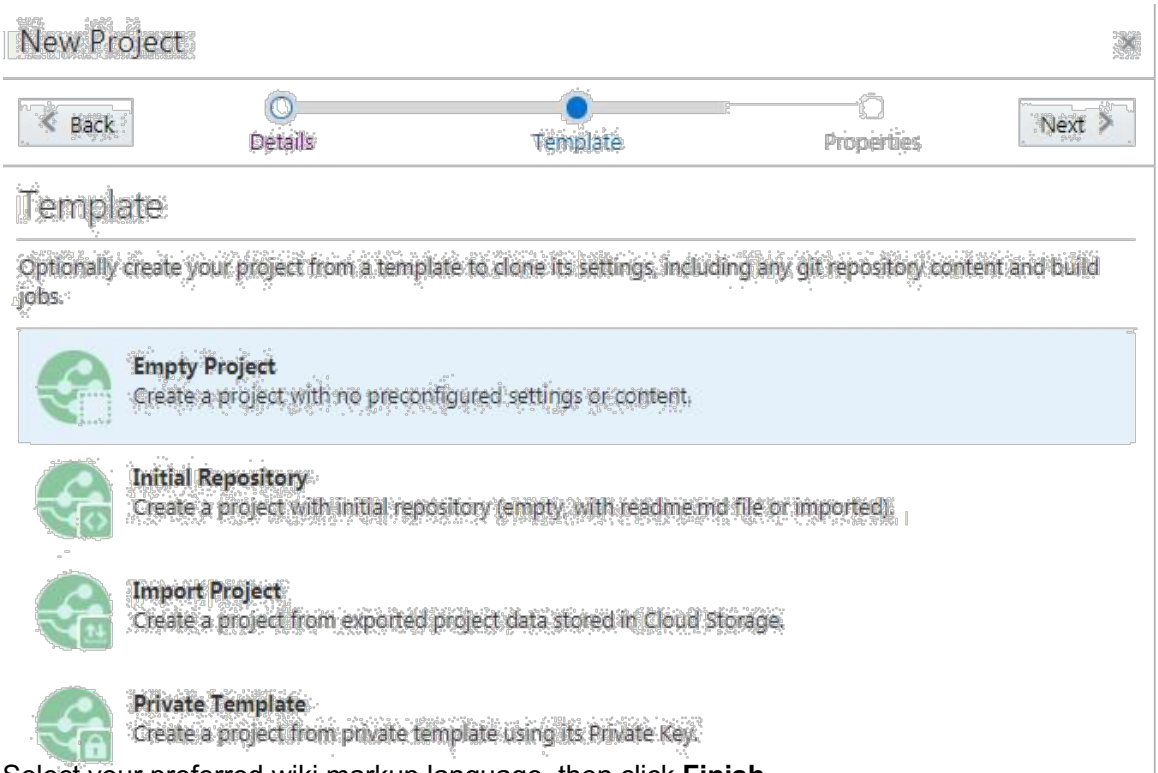
1. Click on the Projects Tab of the Organization page in DevCS.
2. On the Welcome page, click **Create**.

4. Give your project a name that begins with your own name, such as `John Dunbar OOW Hello`, to make your project unique. Then:

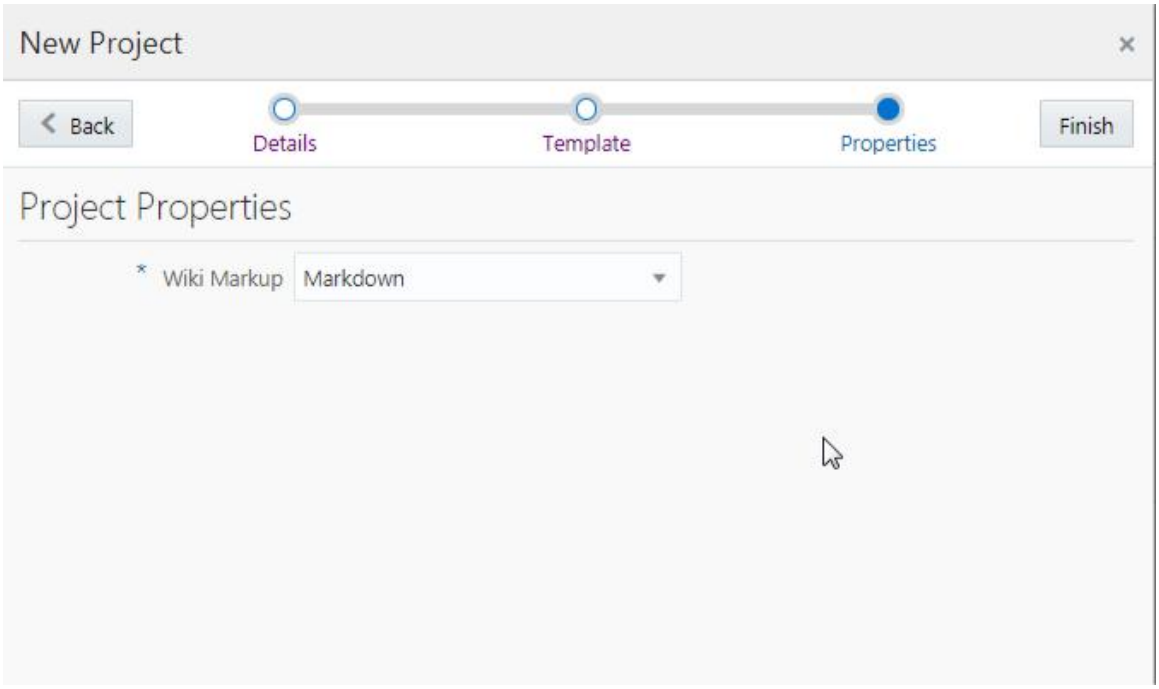
    a. Enter a **Description**, such as `OOW Hello project`.
    b. Leave the Security setting specified as **Private**.
    c. Click **Next**.



5. Click **Empty Project**, then click **Next**

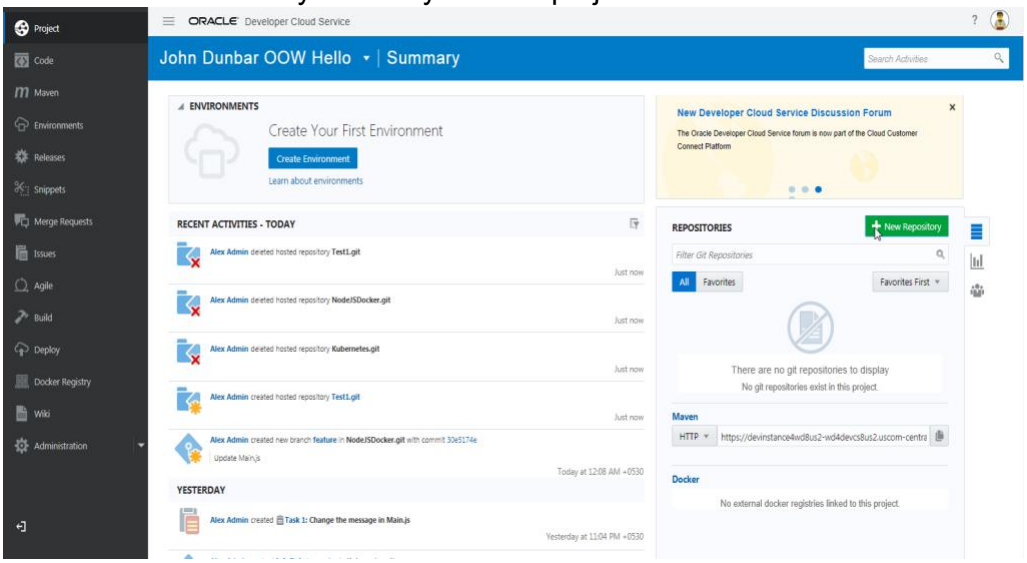


6. Select your preferred wiki markup language, then click **Finish**.



7. Wait while the project modules are provisioned, which can take a minute or two. You can see the indicators turn green as the associated modules are provisioned.

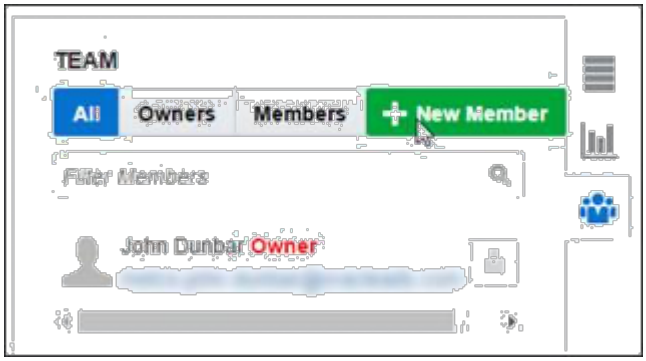
8. When everything is provisioned, the project Home page opens, which contains details about your newly created project:



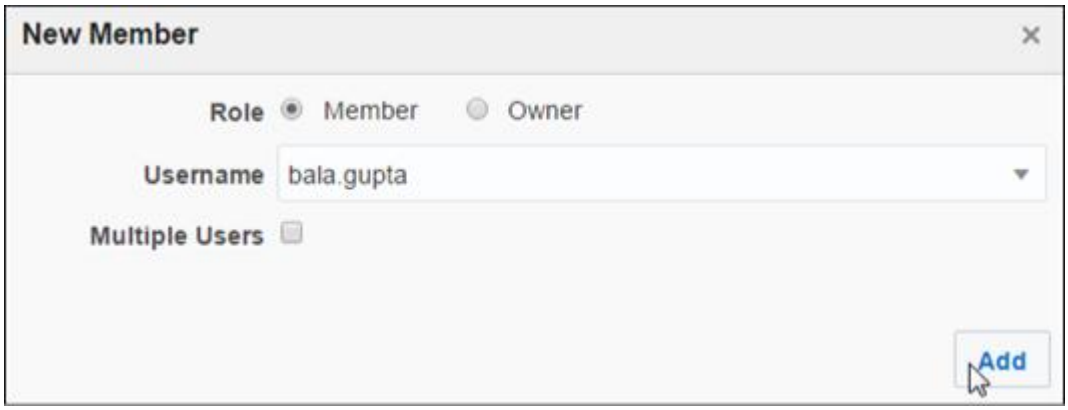
Let's take a look at this page (you may need to scroll to see the whole thing):

- On the left side is an activity feed.
- Tabs on the right side show you where the Git source code and Maven repositories are located.
- Also on the right you can see project statistics, as well as the UI where you can manage team members. Let's take a look at that UI now.

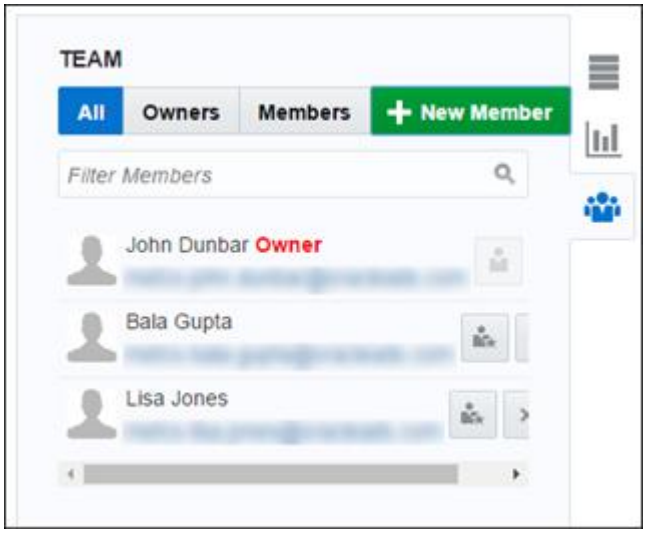
9. Scroll to the right and click the Team tab, then click +**Add Member**.



10. Select another team member from the drop-down list and click **Add**.



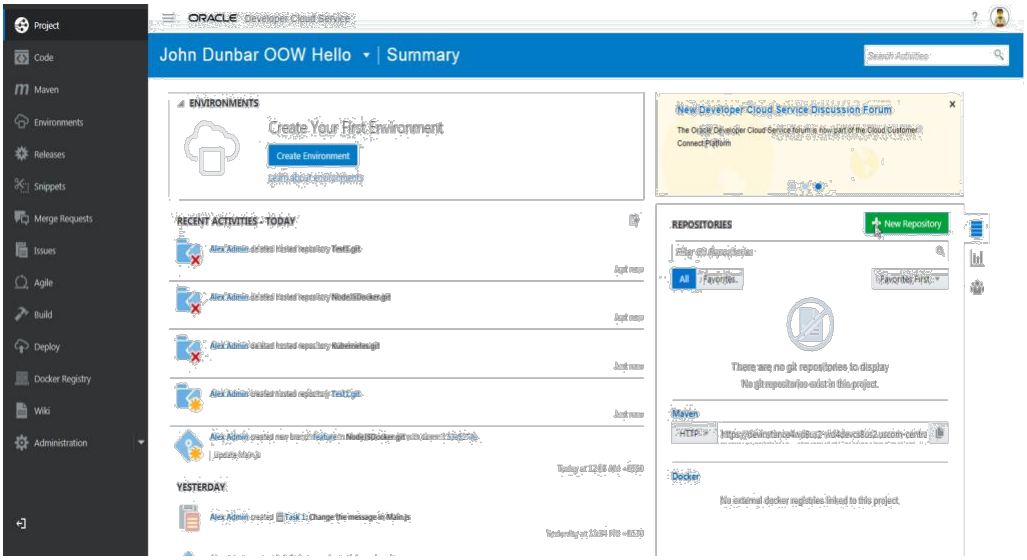
11. Repeat steps 7 and 8 again so that your project team contains two other members besides yourself.



Each team member can now log in to the environment and start collaborating on project development.

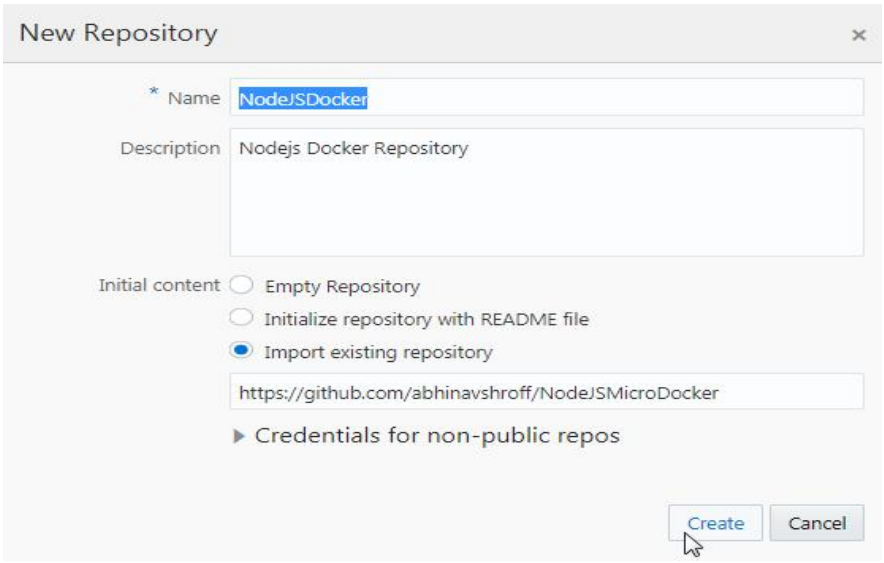
Milestone 2: Fetch and review code from the Git repository

1. On the right side of the home page, click **New Repository**.

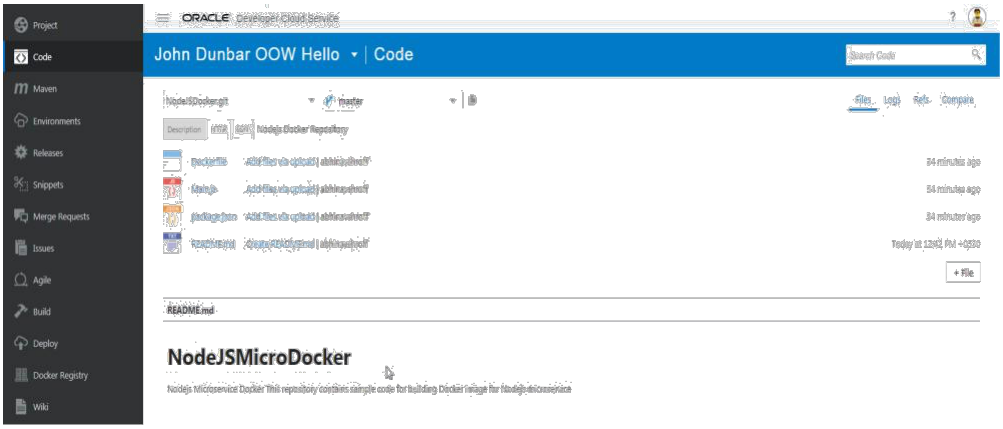


2. In the New Repository dialog, enter these details:

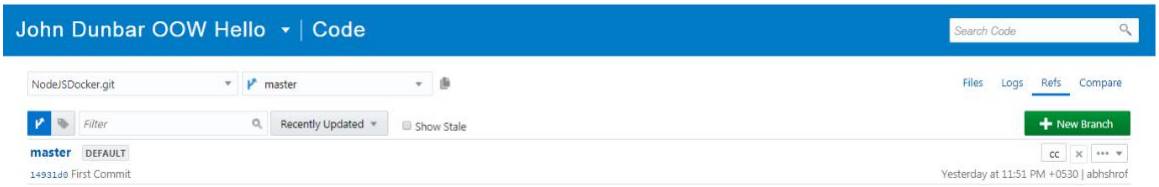
    a. `NodeJSDocker` in the **Name** field
    b. `Nodejs Docker Repository` in the **Description** field
    c. Import Existing Repository under **Initial content**
    d. https://github.com/abhinavshroff/NodeJSMicroDocker in the text box:



3. Click **Create**.
4. You should now be on the Code tab, which shows that you have a new DevCS git repository, NodeJSDocker.git. This new repository contains imported code from the GitHub repository you specified.



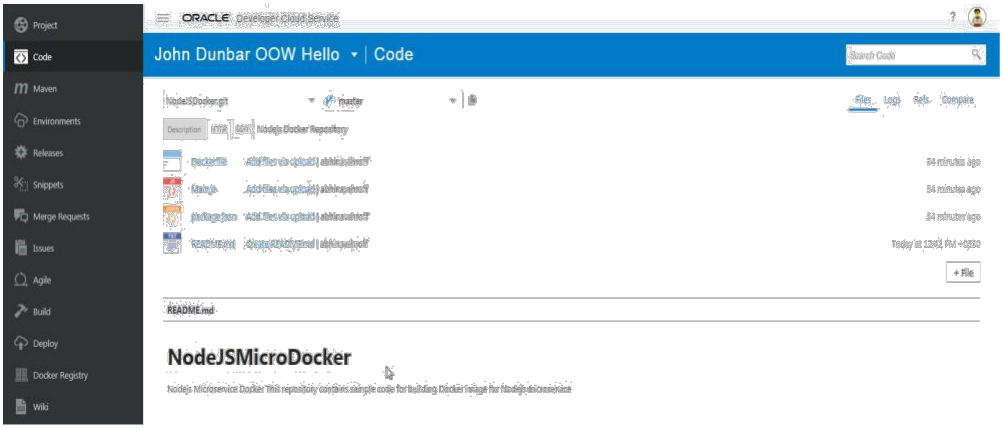
5. Click the **Refs** tab to view the current list of branches. At this point there's only one, the master branch:



6. Click **New Branch**, and create a new branch called `feature`, based on the master branch.

7. Click **Files** in the upper right.

   When you create a new branch, four files are automatically created: Main.js, Dockerfile, package.json, and Readme.md, which describes the repository.



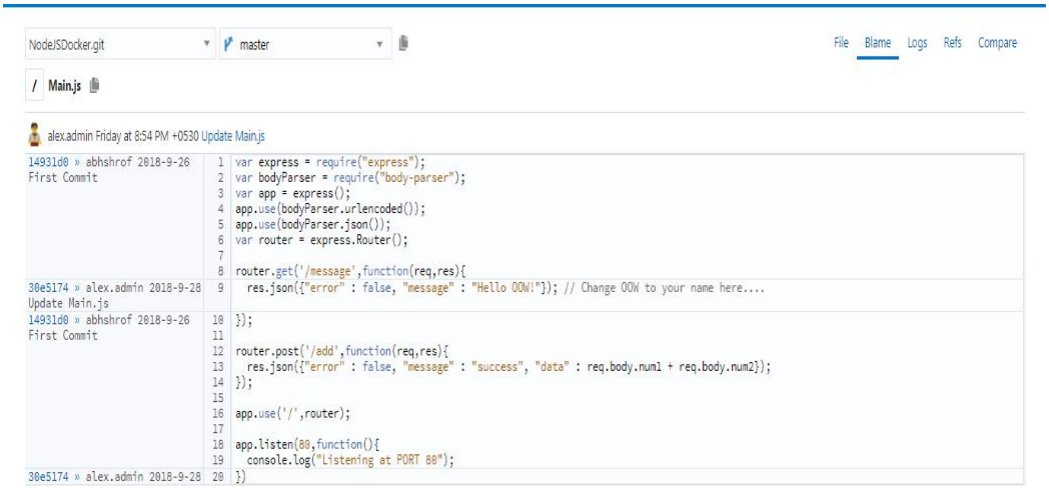
7. Take a look at what's in each file by clicking each file name successively. In particular, notice that:

    - Main.js contains the Node.js REST API code
    - Dockerfile contains the commands to build a Docker image
    - package.json defines the dependencies for the Node.js code

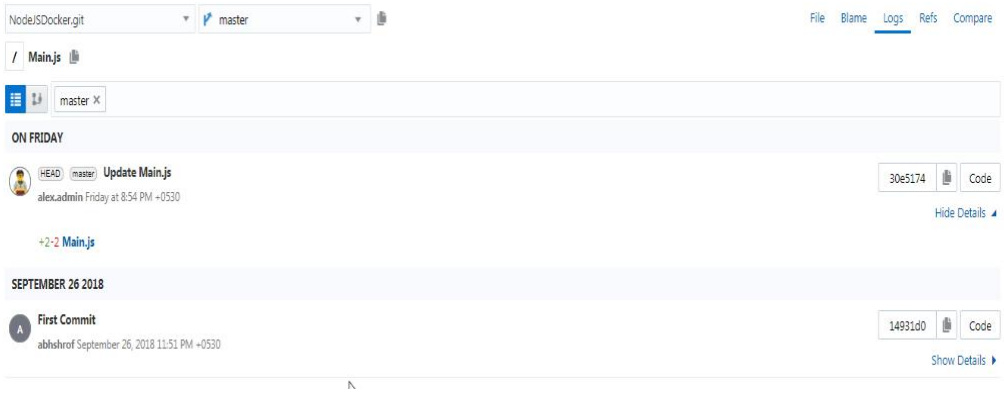
8. Let's go back to Main.js, and on the right notice that there are five methods for viewing a file on the Code tab:

    - File
    - Blame
    - Logs
    - Refs
    - Compare

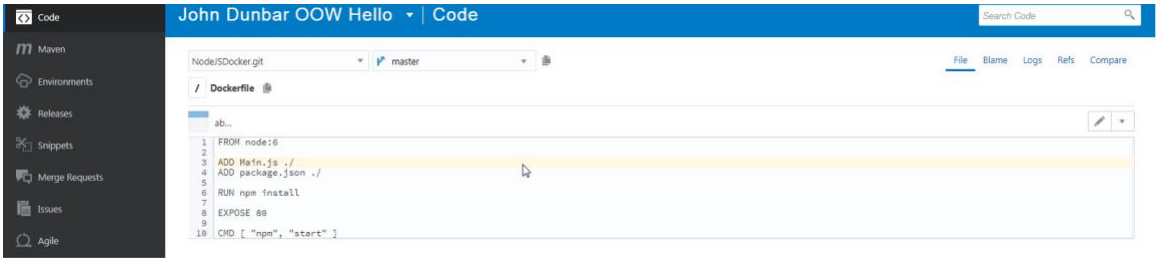
9. Click **Blame** tab to view the code and the file's commit history:



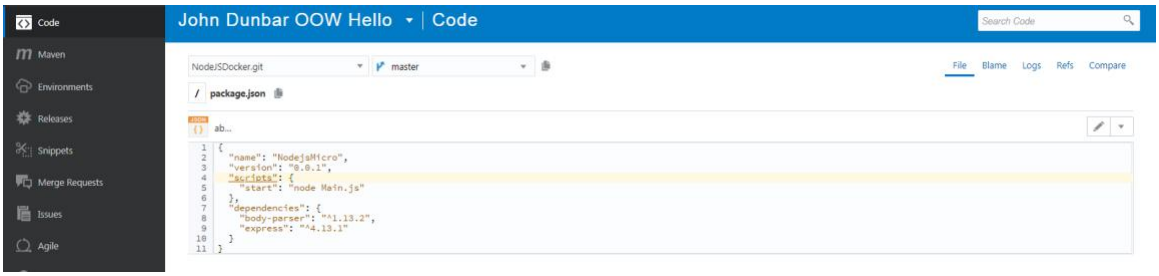
10. Next, click **Logs**, where you can see the list of the commits for this branch, as well as the code that was pushed with each commit:



11. Click **Files**, then click **Dockerfile.** We'll use this file a bit later, when we build the Docker image.



12. Click the forward slash next to the Dockerfile name, then click **package.json** to review the Node.js code dependencies.

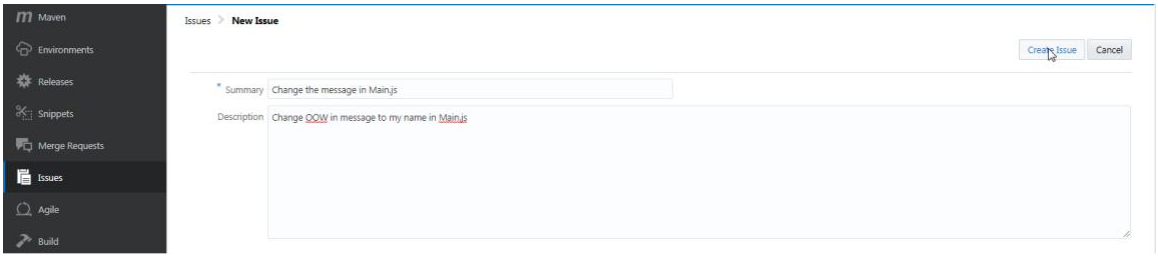


Milestone 3: Track issues for your project

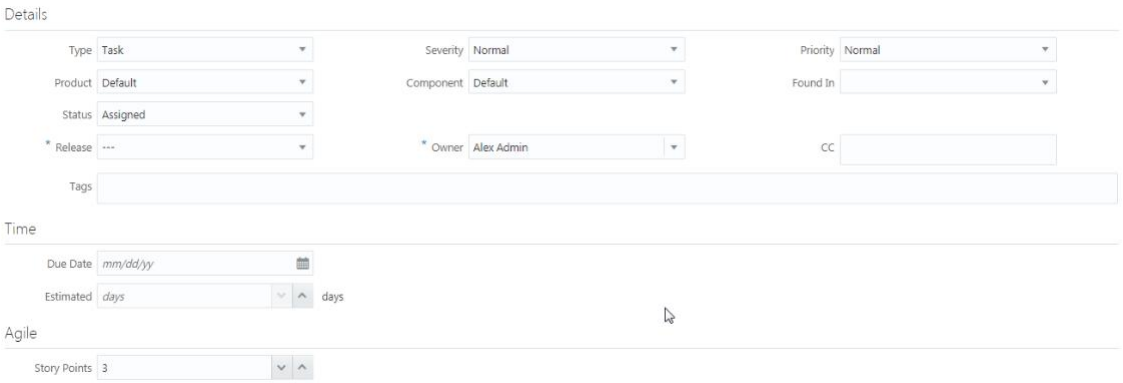
In this section, we'll start tracking tasks (issues) that our team needs to take care of. We'll then organize these tasks into a sprint.

1. In the left nav bar click **Issues**, then click **New Issue**:

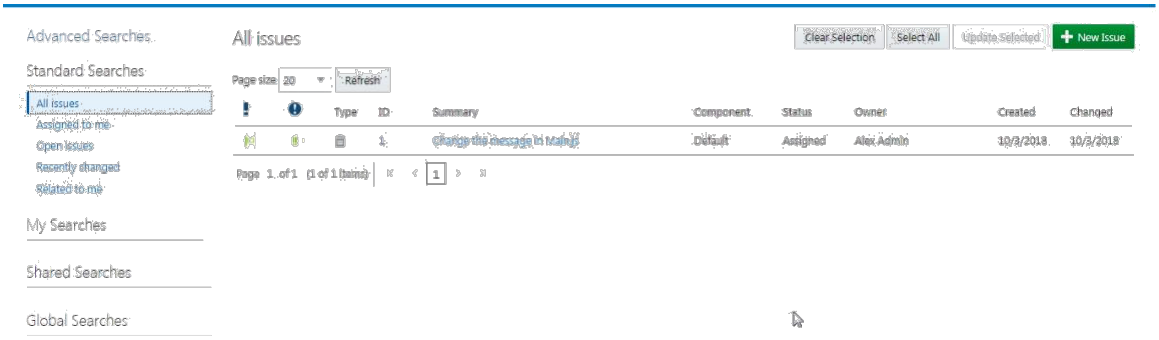
2. For your first issue:
    a. Type `Change the message in Main.js` in the **Summary** field
    b. Type `Change OOW in message to my name in Main.js.` in the **Description** field
    c. Select **Task** in the **Type** drop-down list.



3. Scroll down, and from the **Status** drop-down list, select **Assigned**.
4. From the **Owner** drop-down list, select **kevinevans** (your user).
5. In the **Story Points** field, enter **3**.



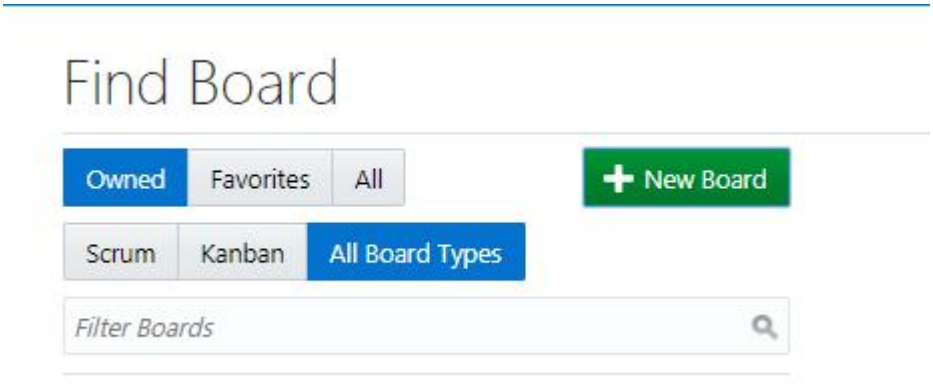
6. Click **Create Issue**.
7. Click the **Issues** tab to see the new issue:



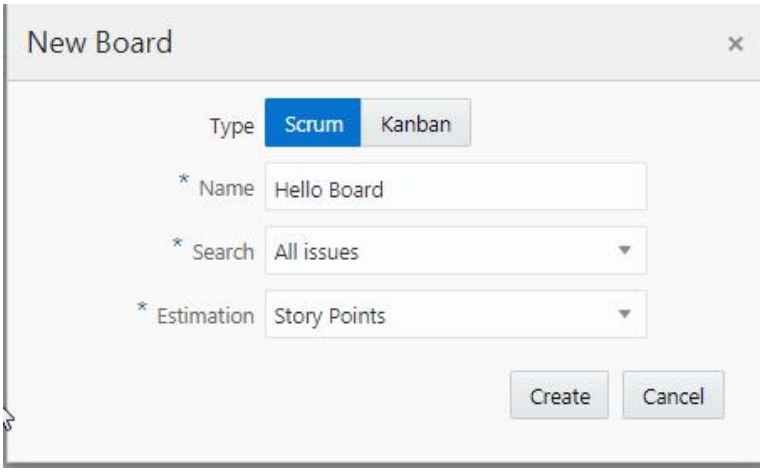
Task 4: Create an Agile board and a sprint to track issues

In this section, we explore the DevCS features that help you manage your agile development process.

1. In the left nav bar, click **Board**, then click **New Board**.

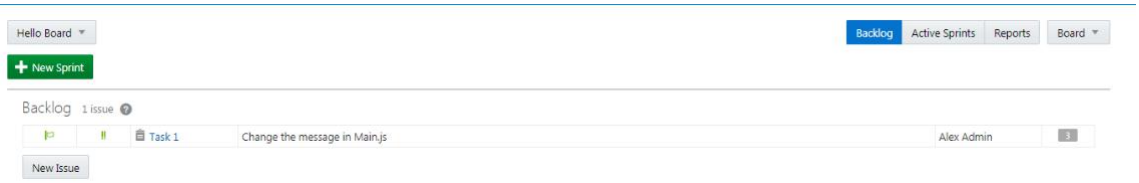


2. In the New Board dialog box:
    a. Type `Hello Board` in the **Name** field
    b. Select **Story Points** from the **Estimation** drop-down list
    c. Click **Create**.

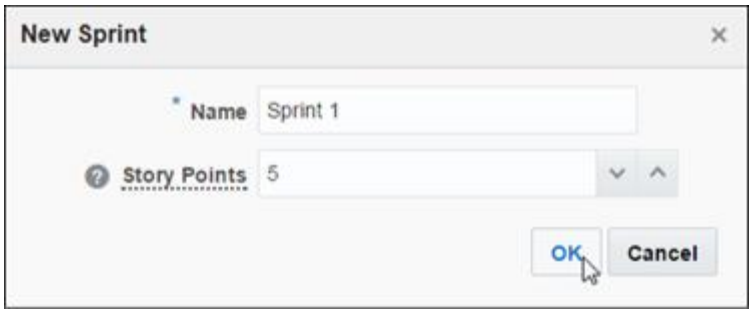


You can see the issue you created earlier in Backlog. Let's add it to a sprint.

3. Click **New Sprint**.

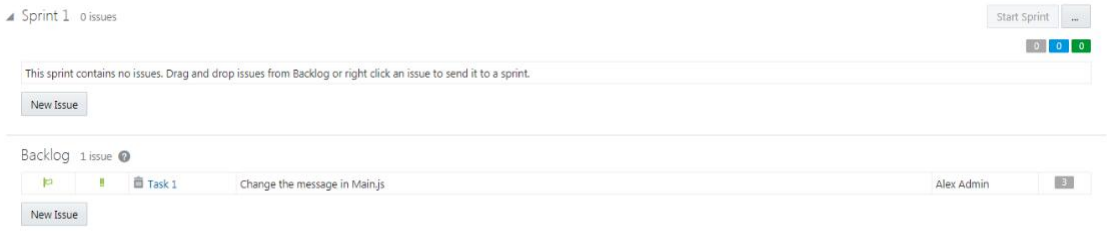


4. Let's call our new sprint `Sprint 1`. In the **Story Points** field enter 5, then click **OK**.



You can now see the new sprint, along with instructions on how to add issues to it.

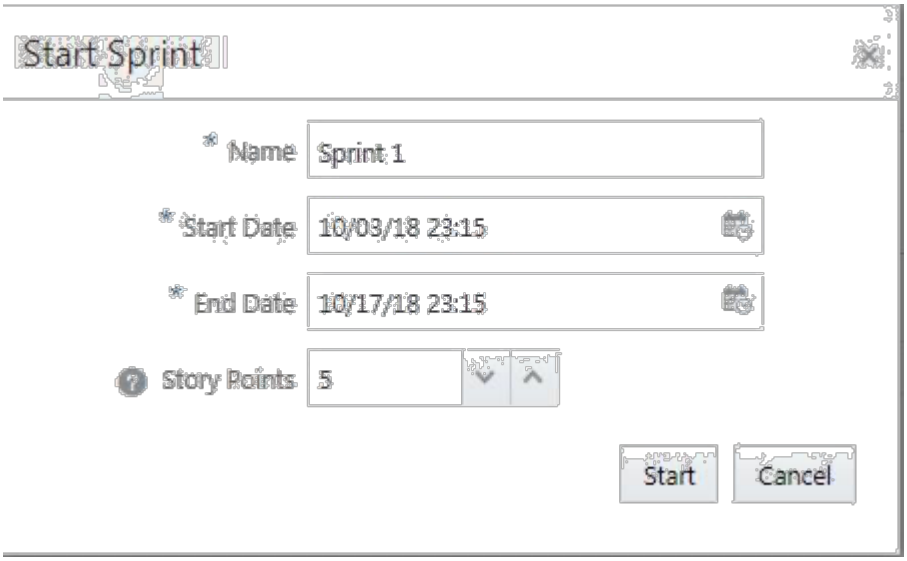
5. Drag and drop Task 1 to Sprint 1, taking care to drop it right on top of the instructions.



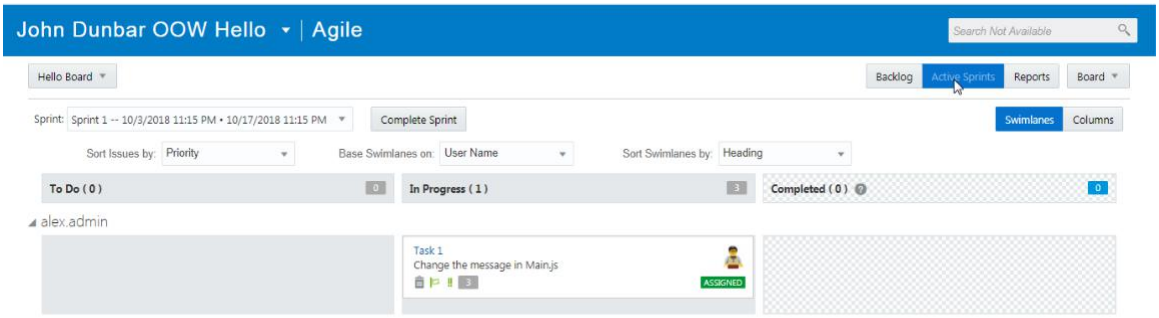
6. Click **Start Sprint** to make the sprint active.

7. In the Start Sprint dialog box, accept the default start and end dates and click **Start**.

   By default, a sprint lasts two weeks. You can now see the default dates above the issue list for Sprint 1.



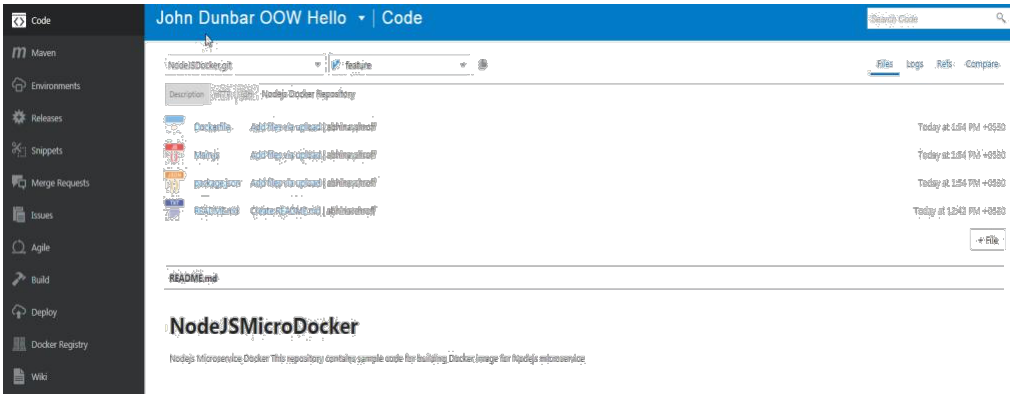
8. Click **Active Sprints** to view the progress of the sprint. You can see at a glance all the tasks considered To Do, In Progress, or Completed.



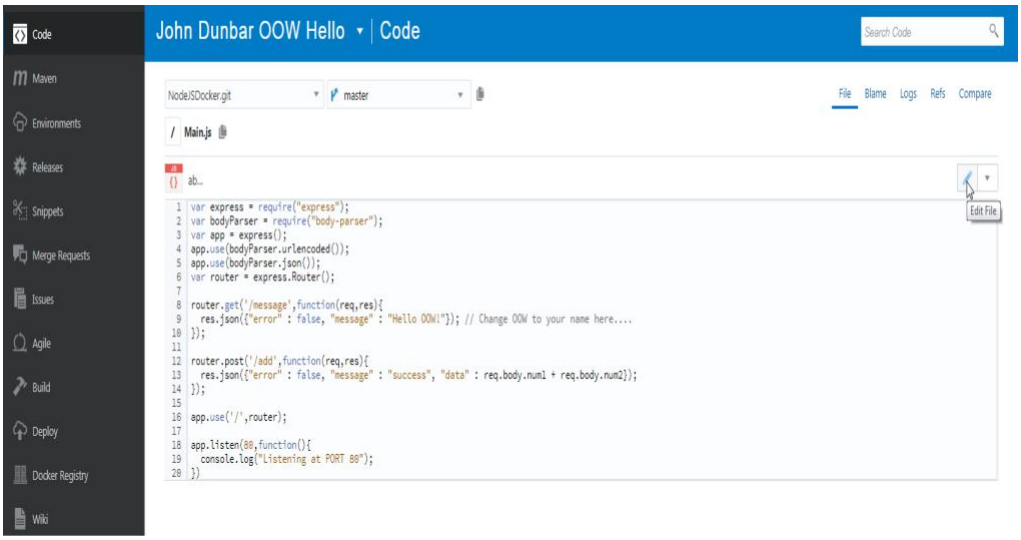
Milestone 5: Edit your code and commit it to the working branch

In this section, we'll see how DevCS helps you edit your code and commit it to the working branch.

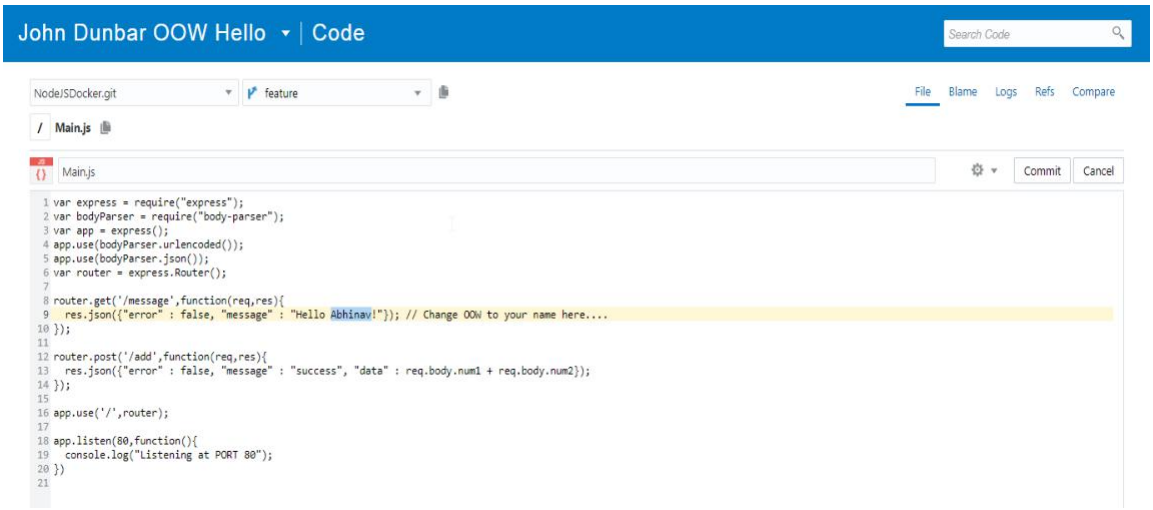
1. Click **Git** in the left nav, then select the NodejsDocker.git repository and the feature branch, as shown here:



2. Click **Main.js,** then click the pencil icon to edit the file.



3. Find where it says “Hello OOW!” and change “OOW” to your own name.



4. Click **Commit**.

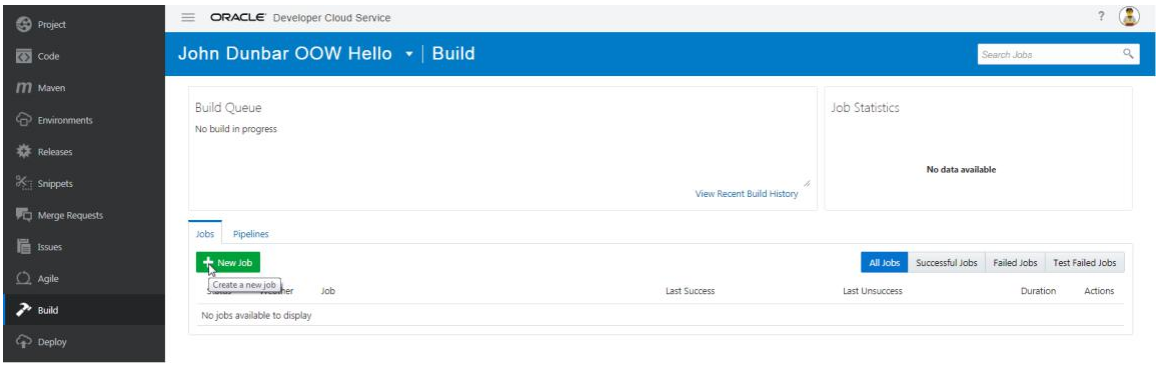
5. In the **Commit changes** dialog, click **Commit** to commit these changes to the NodeJSDocker.git's feature branch:

Milestone 6: Configure the Docker build job

In this section, we'll see how to use DevCS to build the Docker image for the Node.js REST application. We'll actually configure two build jobs, one for NodeJSDockerOCIR and one for NodeJSDockerHub.

Let's start with the NodeJSDockerOCIR build job.

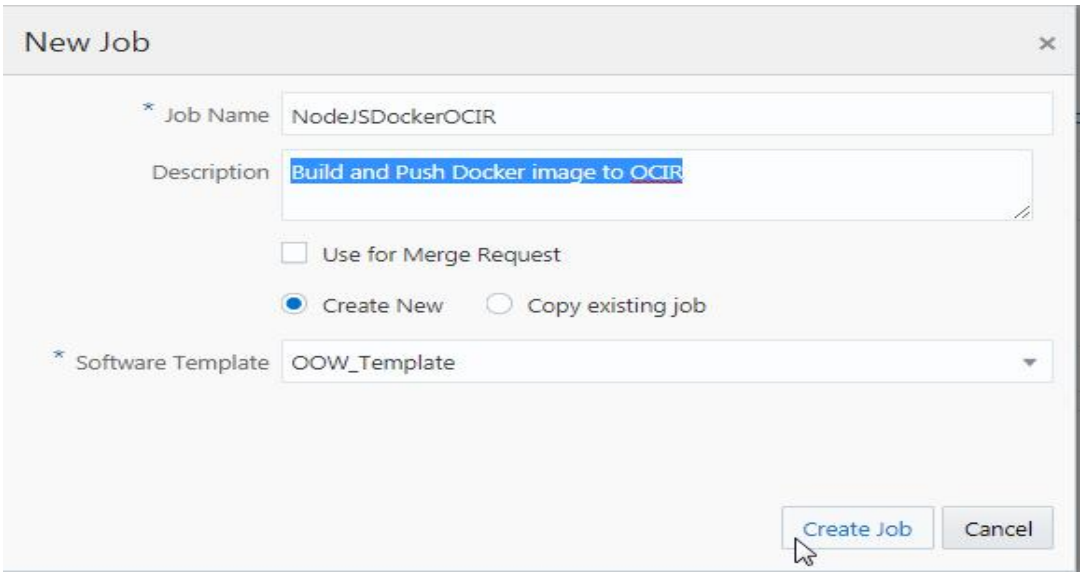
1. Click **Build** in the left nav bar, then click **New Job**.



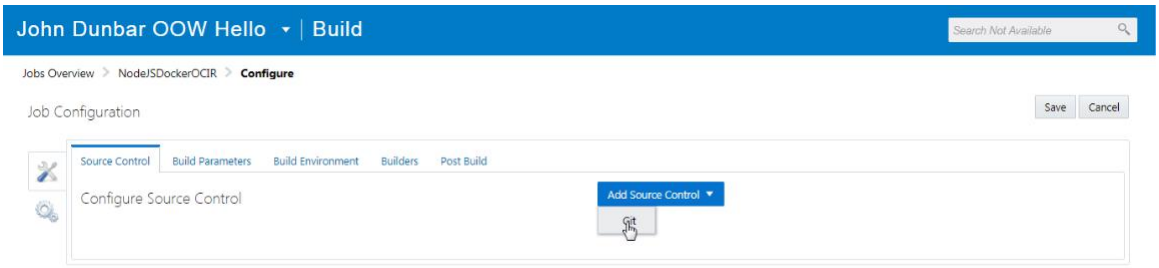
2. In the New Job dialog:

    a. Type `NodeJSDockerOCIR` in the **Job Name** field

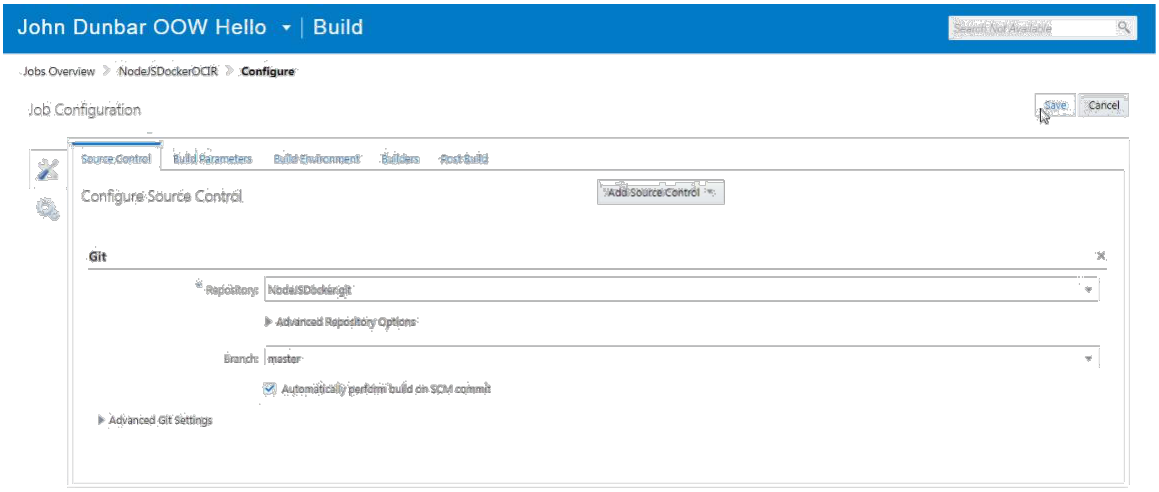
b. Type `Build and push Docker image to OCIR` in the **Description** field
c. Select **DockerTemplate** from the **Software Template** drop-down
d. Click **Create Job**.



3. From the **Add Source Control** dropdown, select Git.

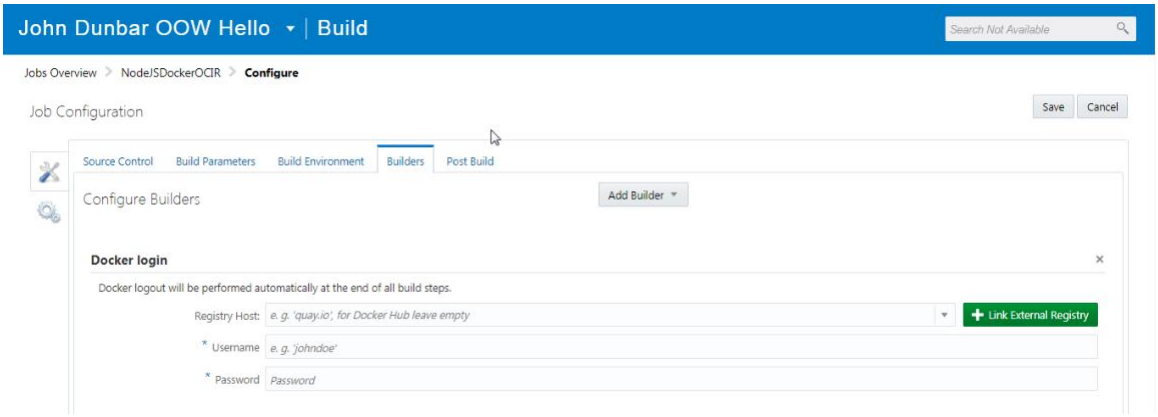


4. Select NodeJSDocker.git from the **Repository** drop-down.
5. Select master from the **Branch** drop-down.
6. Click the **Automatically perform build on SCM commit** check box.



7. Click the **Steps** tab.
8. From the **Add Builder** drop-down, select **Docker Builder->Docker login**.

    a. Type `iad.ocir.io` in the **Registry Host** field
    b. Type `gse00014089/api.user` in the **Username** field
    c. Type `1}UHkK2o[L7nAij7nE{R` in the **Password** field.

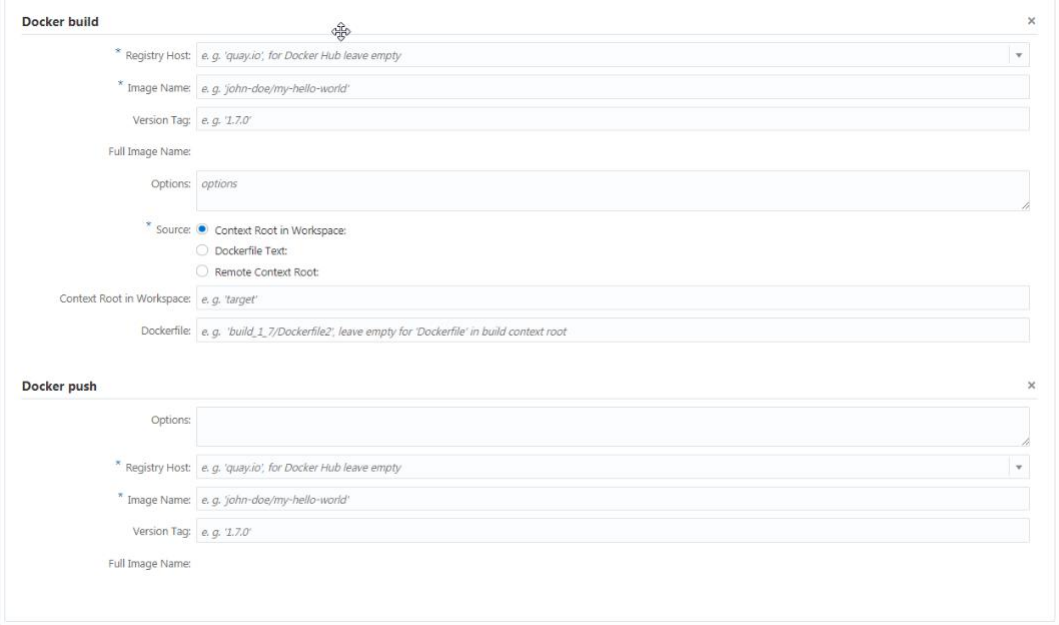


9. From the **Add Builder** drop-down, select **Docker Builder->Docker build**.

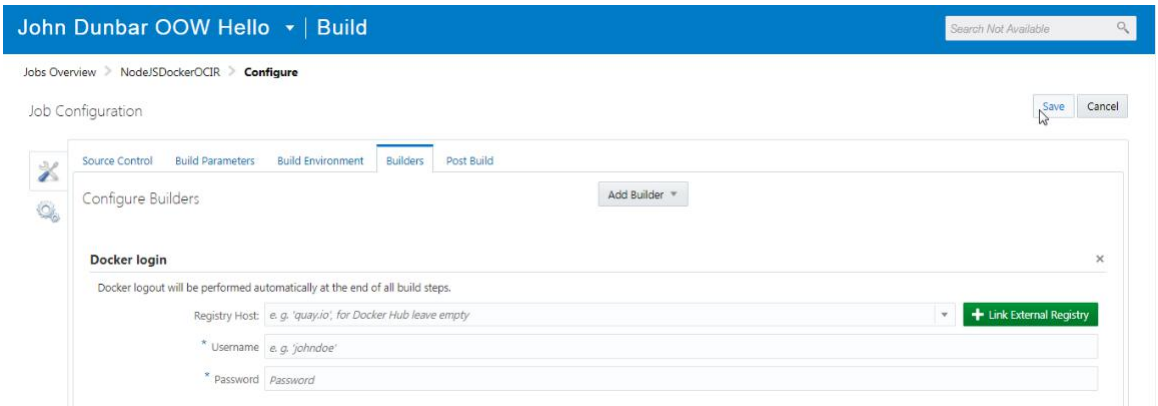
    a. Type `iad.ocir.io` in the **Registry Host** field
    b. Type `gse00014089/oowhol/nodejsmicro`**jodu** in the **Image Name** field. Replace **jodu** with the first two letters of your first and last name, respectively.
    c. In the **Source** radio buttons, click **Context root in Workspace**.

10. From the **Add Builder** drop-down, select **Docker Builder->Docker push**.

    a. Type iad.ocir.io in the **Registry Host** field
    b. Type Type `gse00014089/oowhol/nodejsmicro`**jodu** in the **Image Name** field. Replace **jodu** with the first two letters of your first and last name, respectively.

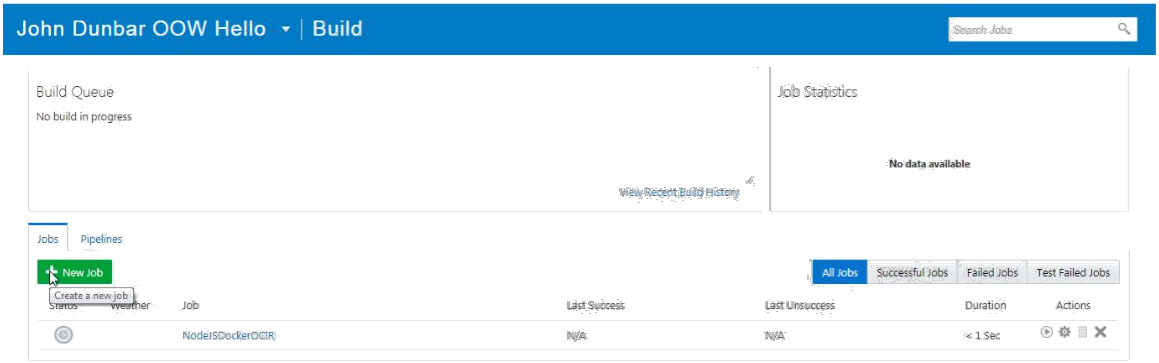


11. Click **Save**.



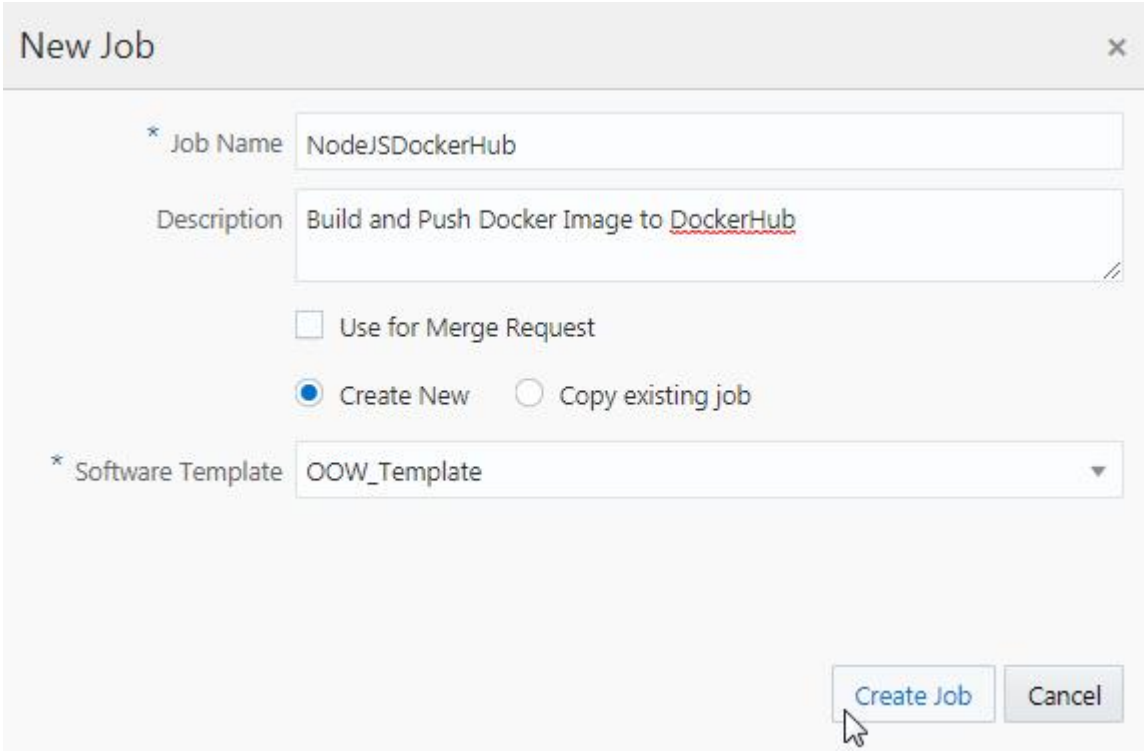
You've finished configuring the first build job, so now let's move on to the NodeJSDockerHub build.

1. Click **Build** in the left nav bar.
2. Click **New Job**.

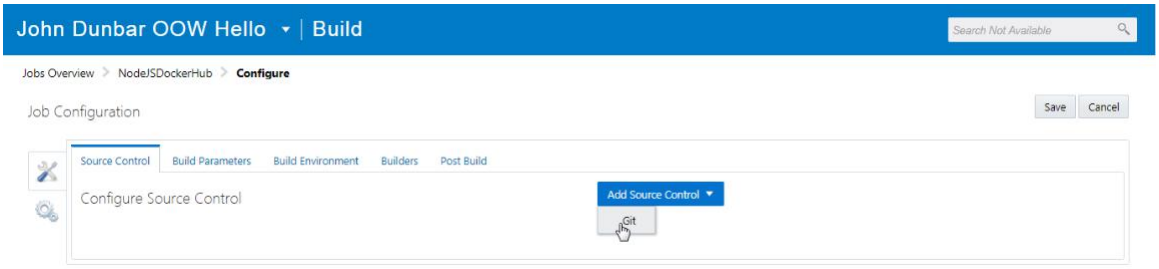


3. In the New Job dialog:

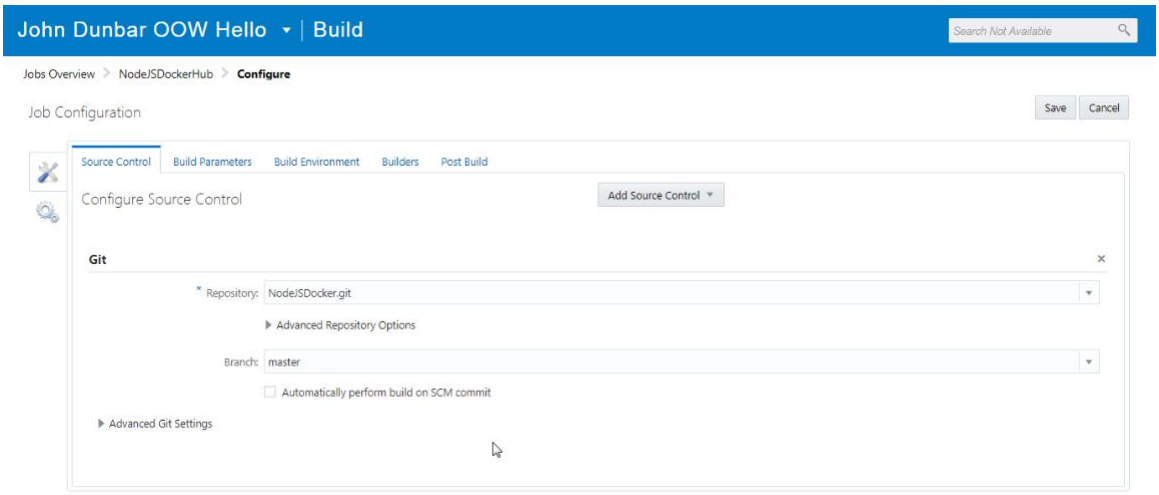
    a. Type `NodeJSDockerHub` in the **Job Name** field
    b. Type `Build and push Docker image to DockerHub` in the **Description** field
    c. Select OOW_Template from the **Software Template** drop-down



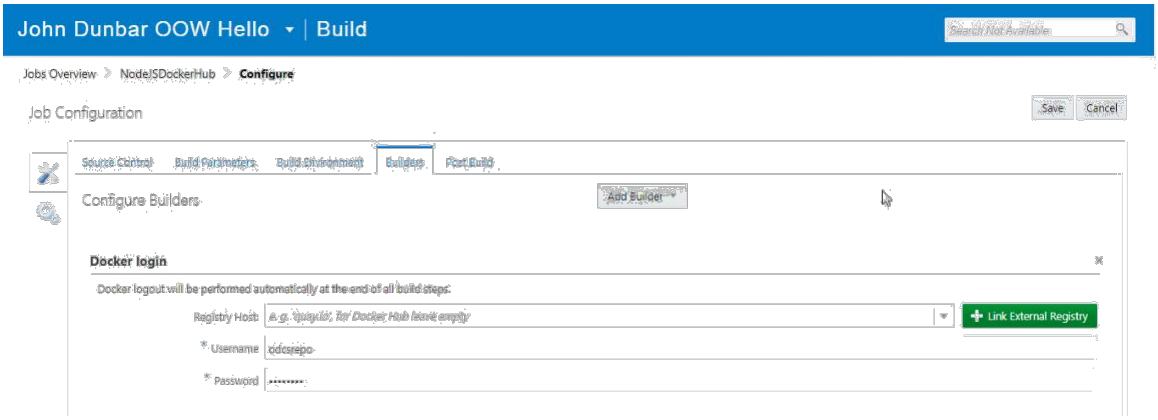
4. Click **Create Job**.
5. From the **Add Source Control** drop-down, select Git.



6. Select NodeJSDocker.git in the **Repository** drop-down.
7. Select master from the **Branch** drop-down.

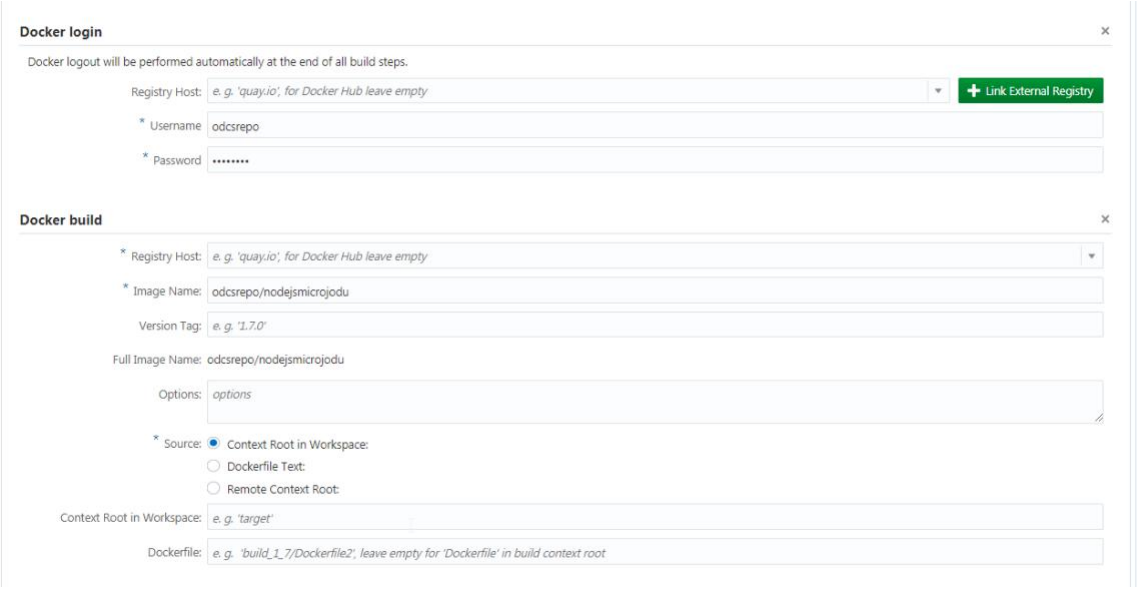


8. Click the **Steps** tab.
9. From the **Add Step** drop-down, select **Docker ->Docker login**.
    a. Type `odcsrepo` in the **Username** field
    b. Type `odcsrepo` in the **Password** field.



10. From the **Add Step** drop-down, select **Docker ->Docker build**.

    a. Type `odcsrepo/nodejsmicro`**jodu** in the **Image Name** field. Replace **jodu** with the first two letters of your first and last name, respectively.
    b. In the **Source** radio buttons, click **Context root in Workspace**.



11. From the **Add Step** drop-down, select **Docker ->Docker Push**.

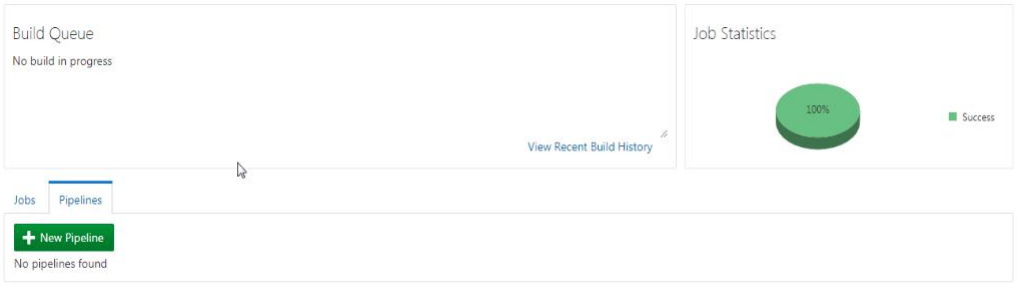
    a. Type Type `odcsrepo/nodejsmicro`**jodu** in the **Image Name** field. Replace **jodu** with the first two letters of your first and last name, respectively.

12. Click **Save**.

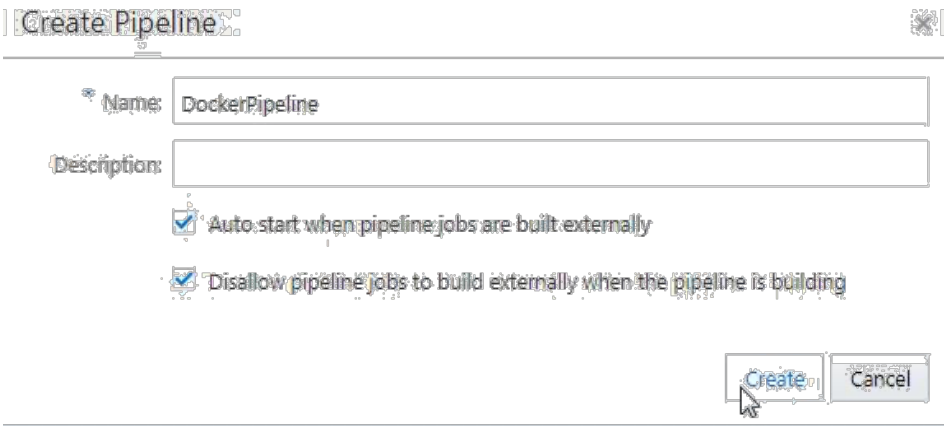
Milestone 7: Create and execute the build pipeline

In this section, we'll create a pipeline to run the two build jobs we just created.

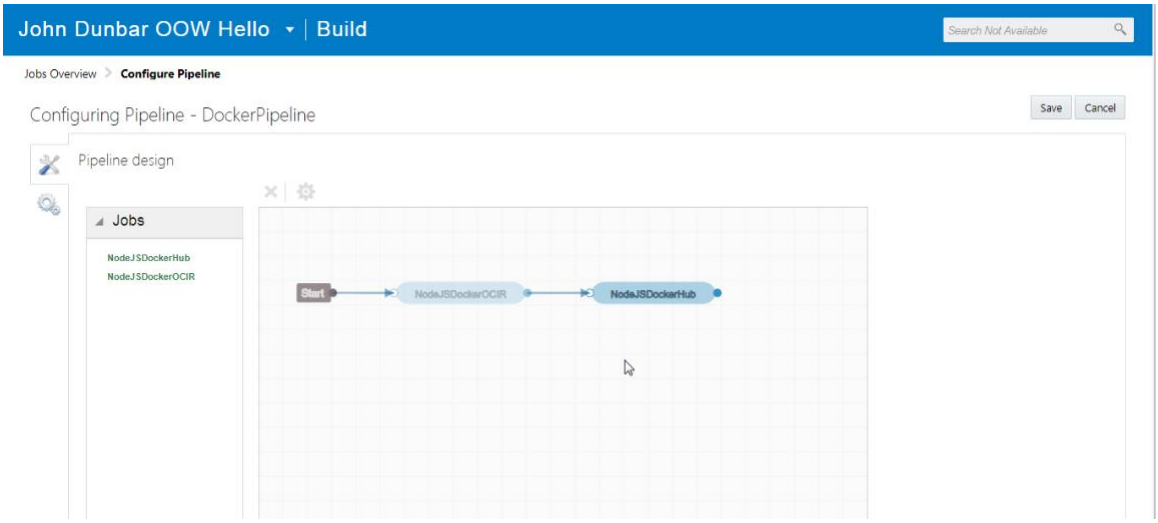
1. Click **Build** in the left nav, then click **Pipelines**:



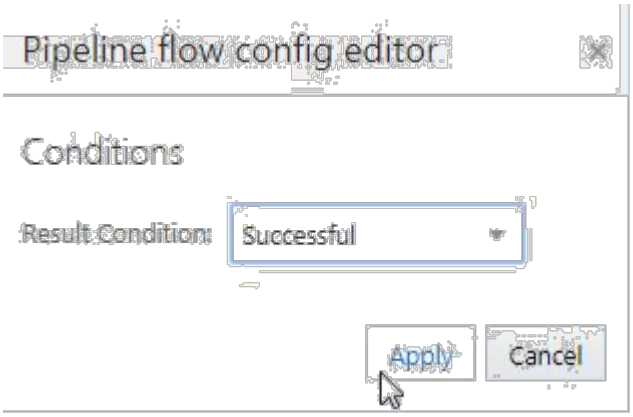
2. Click **New Pipeline**.
3. Let's use **DockerPipeline** as the name.
4. Select both check boxes, as shown here:



5. Click **Create**.
6. Drag NodeJSDockerOCIR to the gridded area, and drop it right after the **Start** bubble.
7. Drag NodeJSDockerHub and drop it right after NodeJSDockerOCIR.
8. User the cursor to draw areas between them, as shown here:



9. Double-click the line connecting the two build jobs.
10. In the editor, select Successful from the **Result Condition** drop-down:



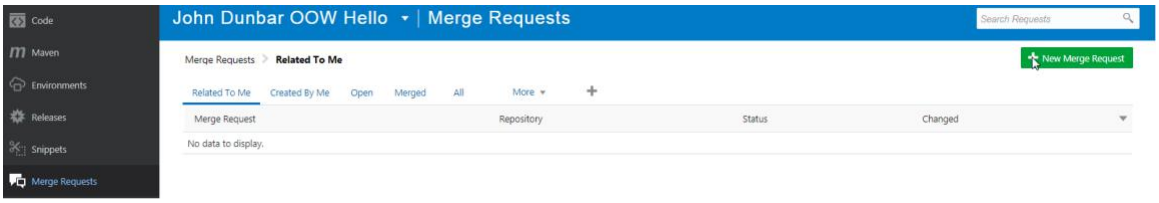
11. Click **Apply**.
12. Click **Save**:

Milestone 8: Merge your feature branch with the master branch of the NodeJSDocker.git repository

In this section, we explore how to review and merge code in DevCS.

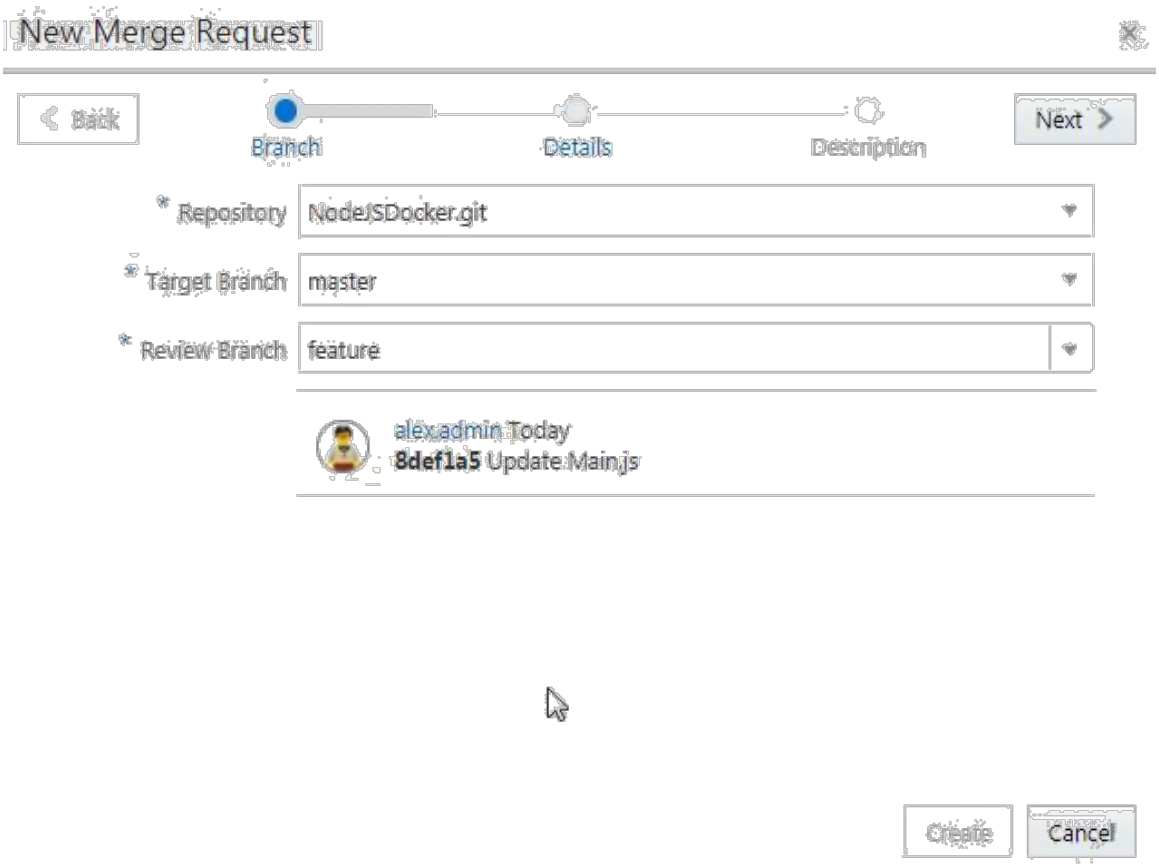
1. Click **Merge Requests** in the left nav bar:

2. Click **New Merge Request**:



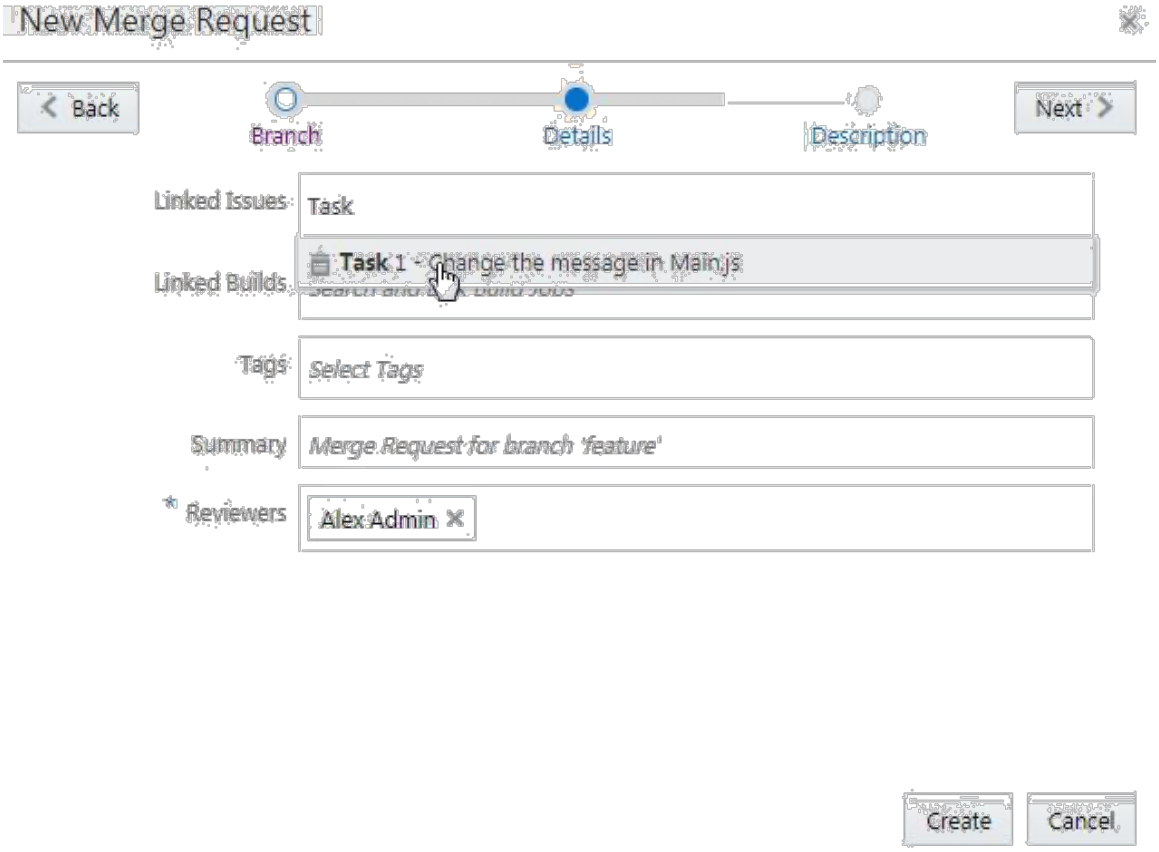
3. In the New Merge Request dialog:

    a. Select NodeJsDocker.git from the **Repository** drop-down
    b. Select master as the **Target Branch**
    c. Select feature as the **Review Branch**
    d. Click the commit that appears for the review branch
    e. Click **Next**.

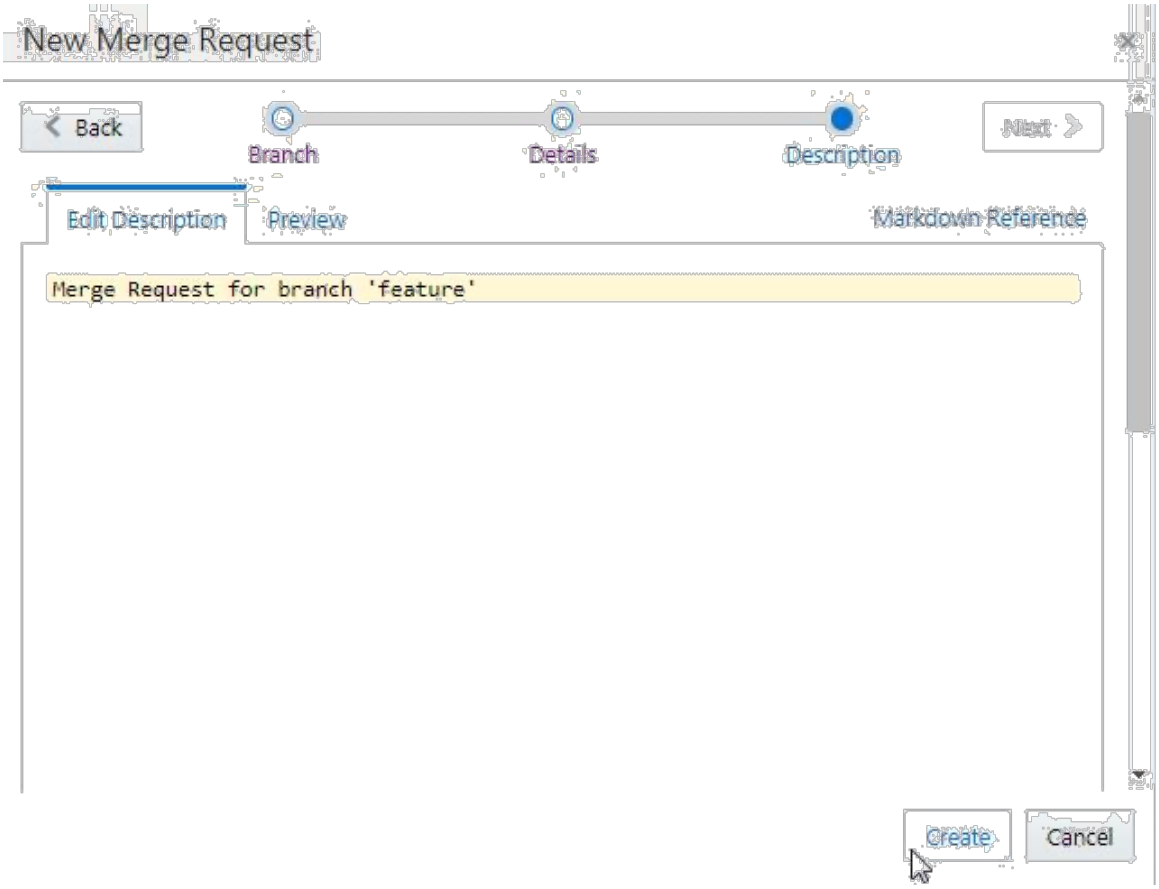


4. On the Details page:

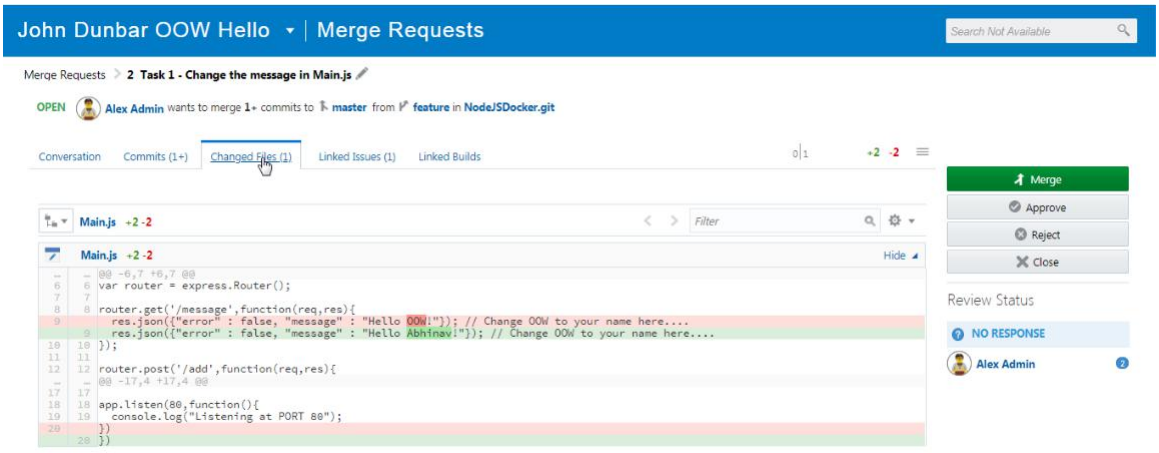
    a. Select yourself as the reviewer under **Reviewers**
    b. In the **Linked Issues** field, type Task until you see the task you created earlier.
    c. Click the task description
    d. Click **Next**.



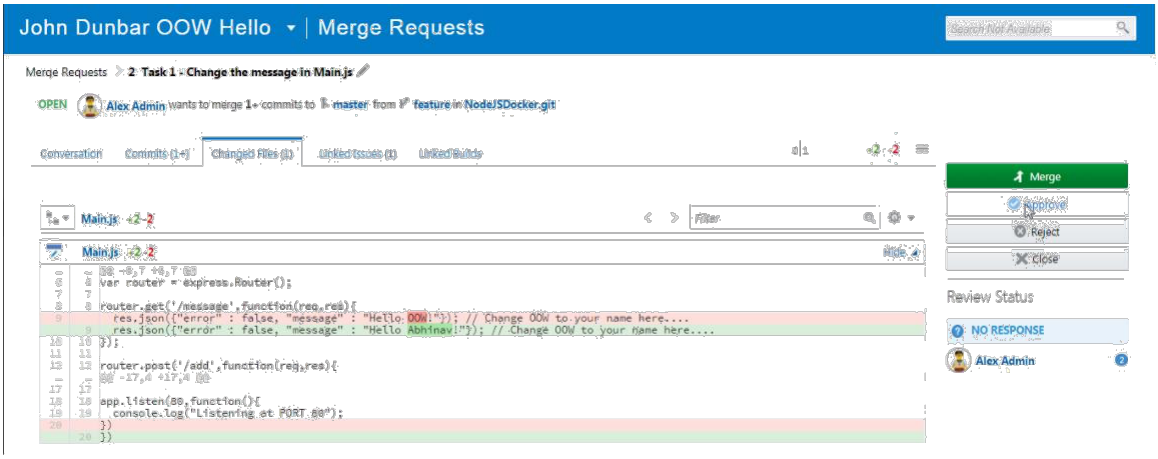
5. Review the description and click **Create**:



6. Click the **Changed Files** tab and review the changes:

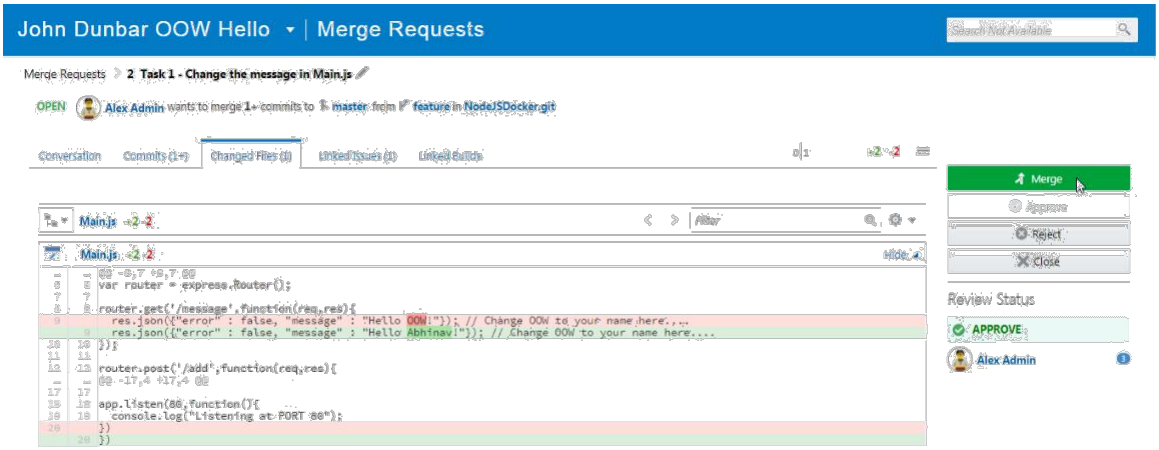


7. If the changes look correct, click **Approve**:

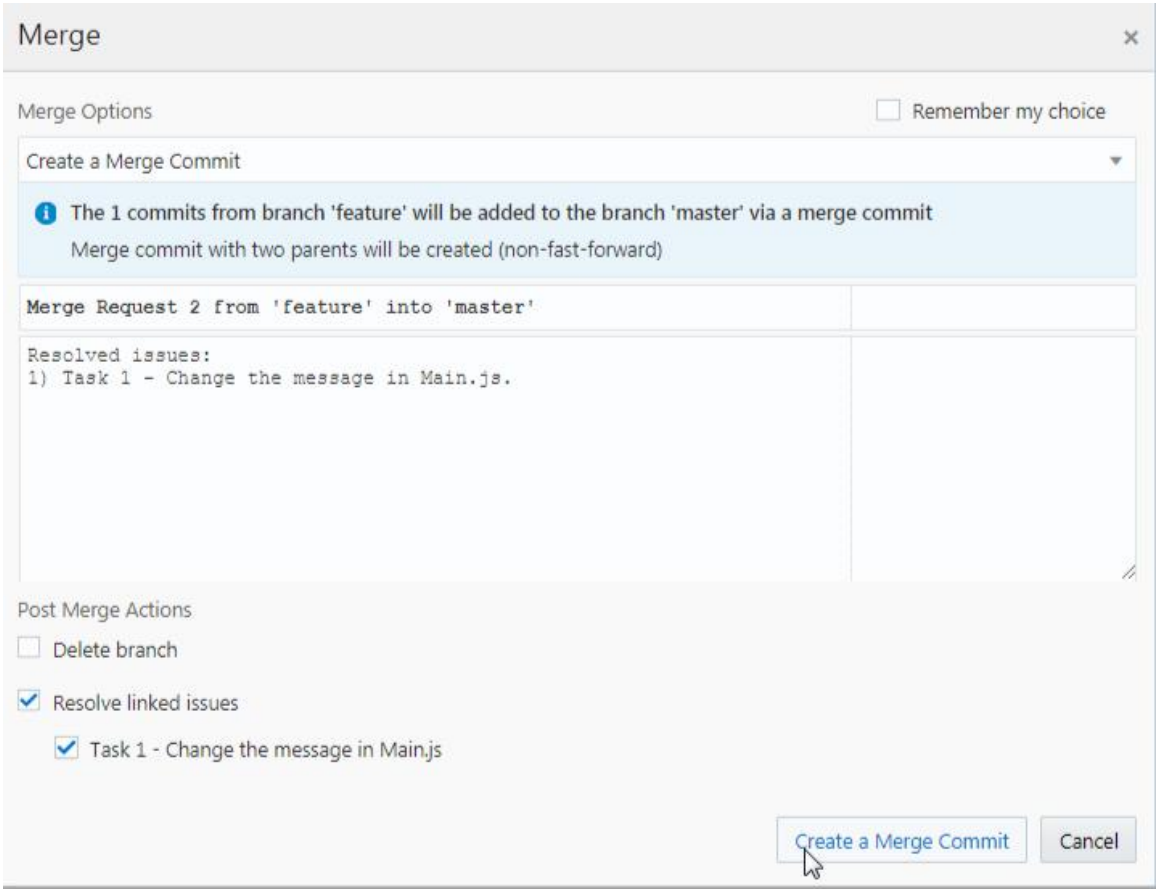


8. Enter OK as a comment, then click **OK**:

9. Click **Merge** to merge the changes into the NodeJSDocker.git repository's master branch:

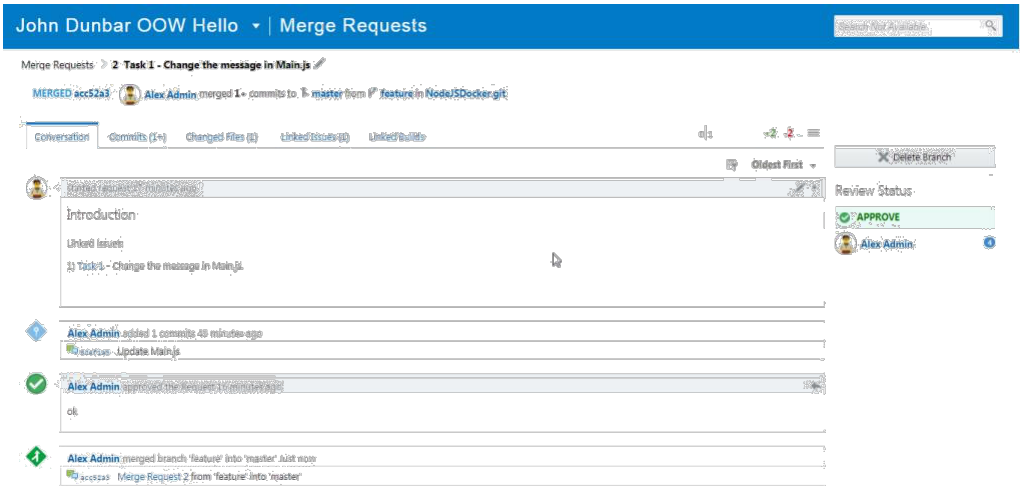


10. Keep the defaults, then click **Create a Merge Commit**.

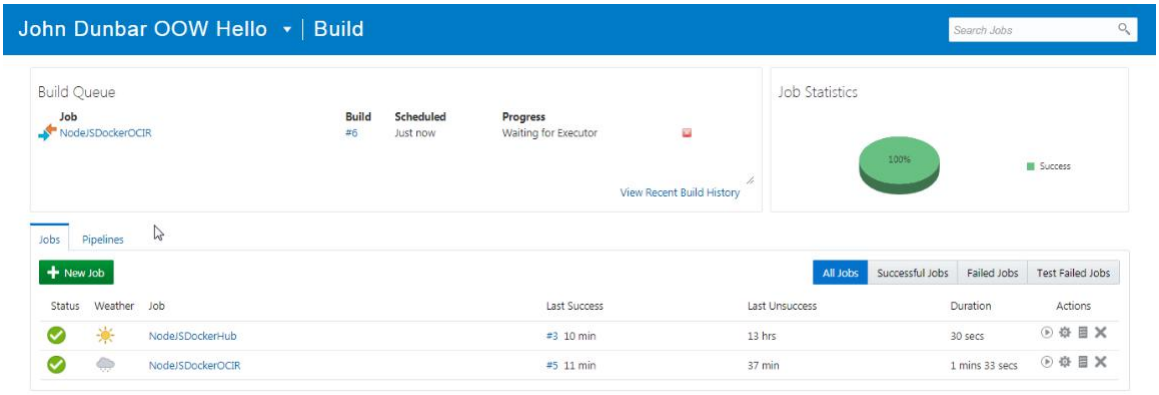


You should now see the Review Status as Approve, with a green tick.

This merge will trigger the execution of the **NodeJSDockerOCIR** build job, which in turn will trigger the execution of the **DockerPipeline.**

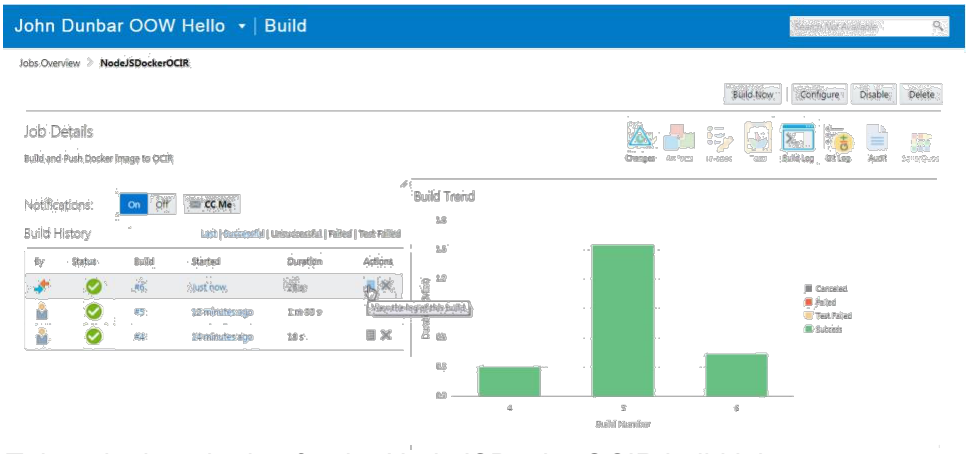


11. Click **Build** in the left nav bar.
12. Click **Jobs**:



You should see your two build jobs, waiting to be built.

13. Click the **NodeJSDockerOCIR** build job.
14. Click **Build Log**, as shown here:

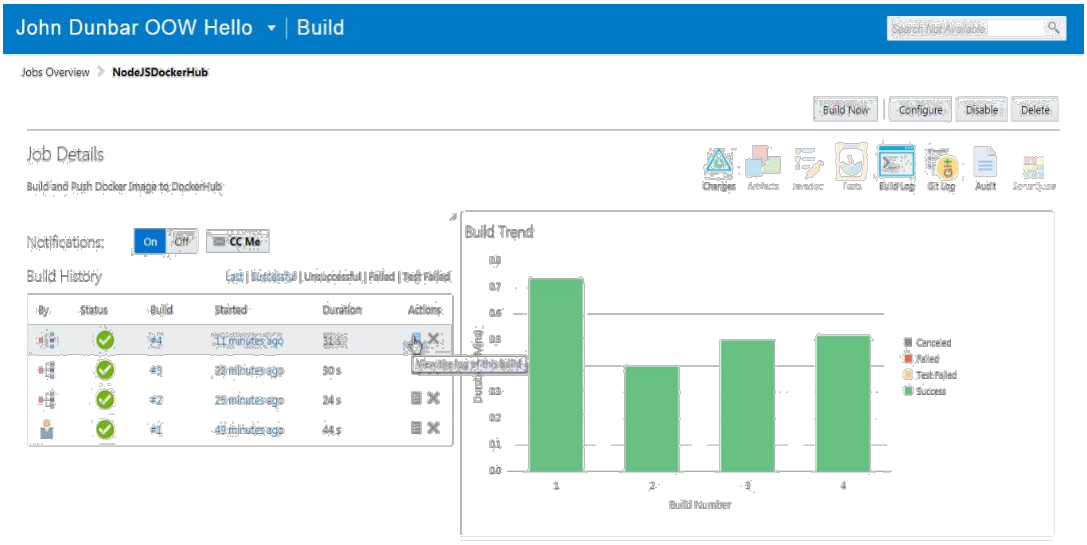


15. Take a look at the log for the NodeJSDockerOCIR build job.

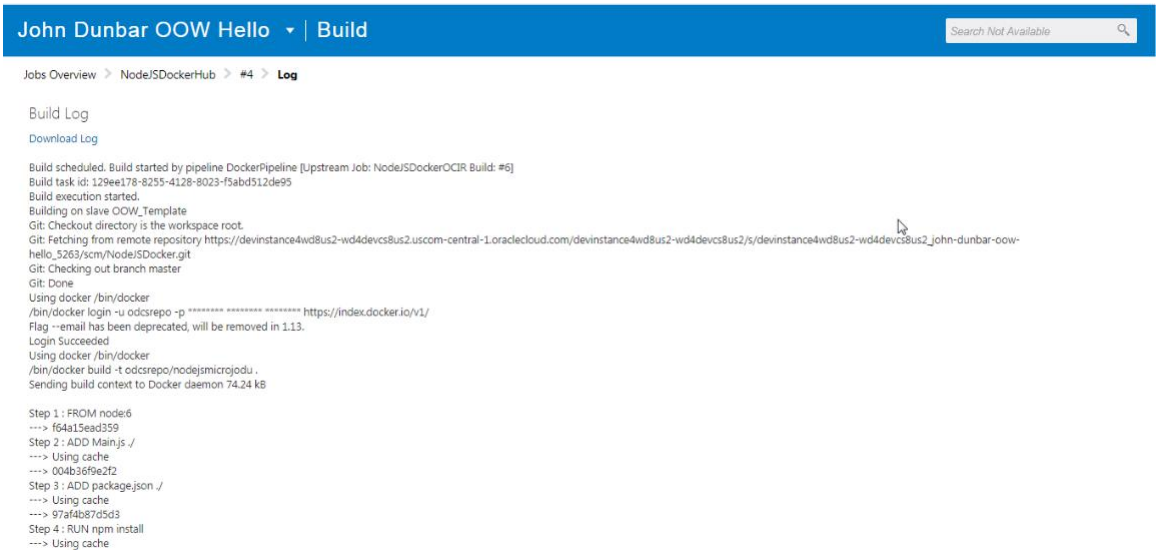
```
---> 404cf4a823ae
Successfully built 404cf4a823ae
Using docker /bin/docker
/bin/docker push iad.ocir.io/devcstest/oowhol/nodejsmicrojodu
The push refers to a repository [iad.ocir.io/devcstest/oowhol/nodejsmicrojodu]
db41ee45fdb9: Preparing
74371bfbfe3e: Preparing
2343c7967ba4: Preparing
6c9a1bc6b767: Preparing
f059848e1802: Preparing
2793dc0607dd: Preparing
74800c25aa8c: Preparing
ba504a540674: Preparing
81101ce649d5: Preparing
daf45b2cad9a: Preparing
8c466bf4ca6f: Preparing
2793dc0607dd: Waiting
74800c25aa8c: Waiting
ba504a540674: Waiting
81101ce649d5: Waiting
daf45b2cad9a: Waiting
8c466bf4ca6f: Waiting
74371bfbfe3e: Layer already exists
f059848e1802: Layer already exists
db41ee45fdb9: Layer already exists
6c9a1bc6b767: Layer already exists
2793dc0607dd: Layer already exists
ba504a540674: Layer already exists
74800c25aa8c: Layer already exists
81101ce649d5: Layer already exists
daf45b2cad9a: Layer already exists
8c466bf4ca6f: Layer already exists
2343c7967ba4: Layer already exists
latest: digest: sha256:31b304cc366a64f48032282918dc7c9e5a8b670db18463cbea2b8d46860447dd size: 2632
/bin/docker logout iad.ocir.io
Remove login credentials for iad.ocir.io
Slave log size 2.1 KB (2,141)

Build completed.
Status: DONE Result: SUCCESSFUL Duration: 26 sec
```

16. Click the NodeJSDockerHub build job and take a look at its build log too:



17. Inspect the build job:



18. Open Google Chrome and go

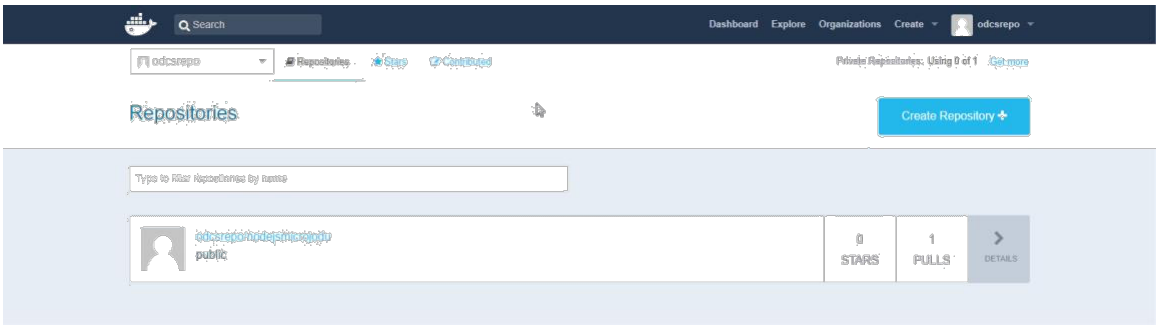
    to: https://hub.docker.com

19. Sign in using these credentials:

    Username: odcsrepo

    Password: odcsrepo

You should see your Docker image in the repository!



Summary

Now that you've finished this lab, you should know how to build a pipeline to build and push a Docker image to Oracle Cloud Infrastructure Registry and DockerHub registry using Developer Cloud. Specifically, you learned how to:

- Work with Oracle Developer Cloud Service and its most important features
- Create a project and pull the code from an existing GitHub repository to a DevCS-hosted Git repository
- Create a task within DevCS and an Agile board to track it
- Complete the task by branching the code, editing it using the online editor, and merging the code
- Build the Docker image with Continuous Integration in DevCS
- Push the Docker image to the Oracle Cloud Infrastructure Registry and DockerHub.