Low Code Development for Production

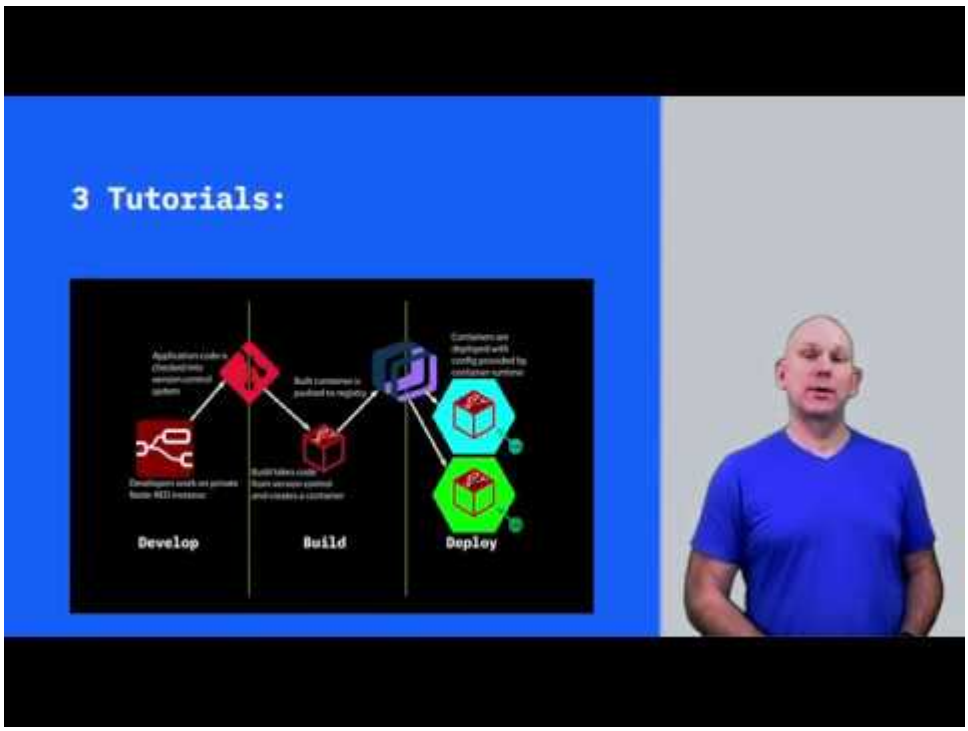
How to integrate Low Code development into a production development pipeline

Brian Innes

Table of contents

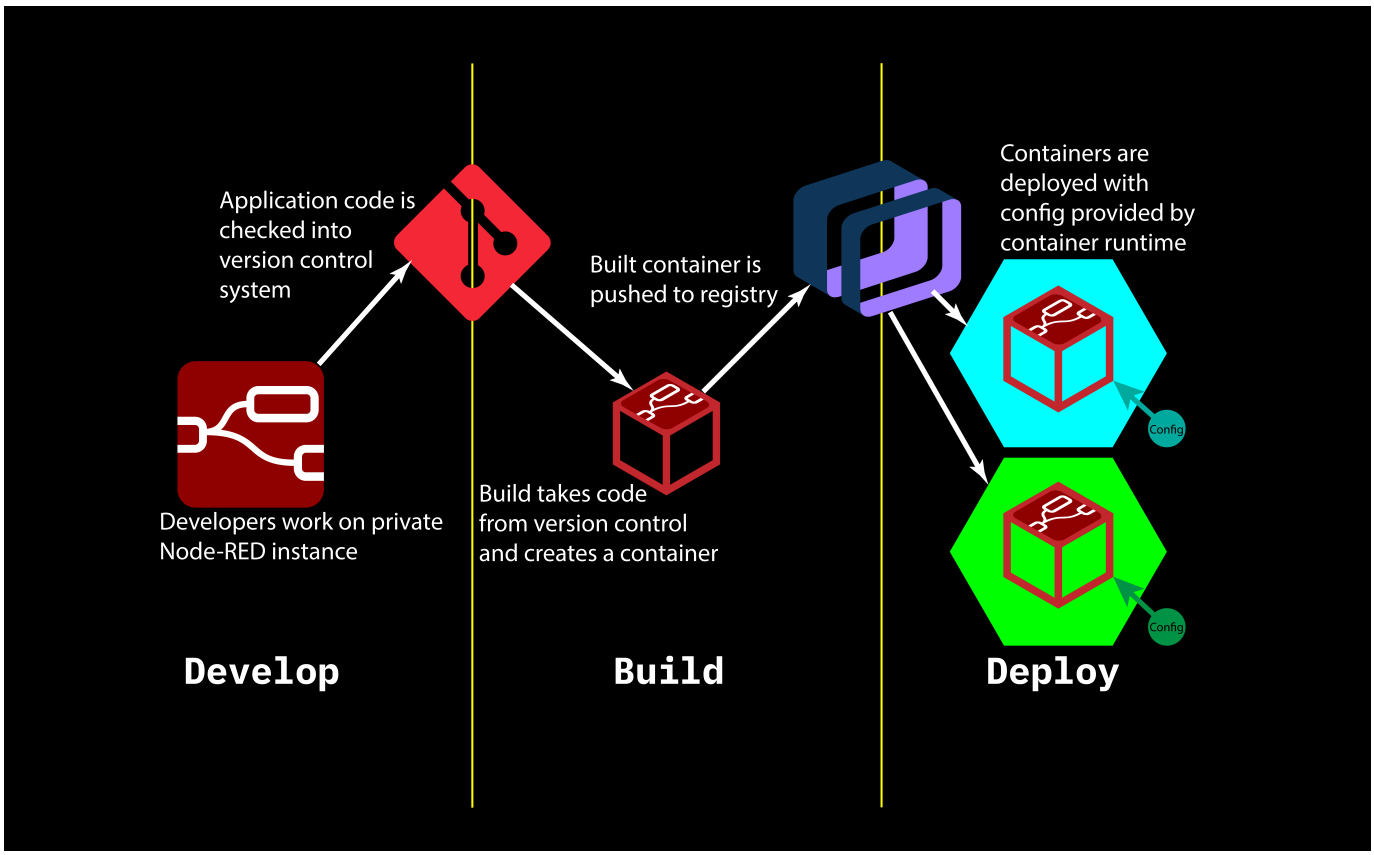
1. Low Code Development with Node-RED for Production

Late in 2019 Node-RED released version 1.0. In the following video the use of Low Code development for production use cases is discussed.



One of the reported difficulties with using Node-RED in production is how to integrate Node-RED into a DevOps process. Here are a series of Tutorials to show how to integrate Node-RED into a dev ops pipeline and also the considerations that need to be made when creating the application, so it can be deployed to a cloud environment.

To use Node-RED in a DevOps pipeline, the development process needs to looks like other programming languages:



1. Developer works locally to create applications
2. Code is delivered to a version control system, such as Git
3. A build process creates the application from source, packages it as a container and stores it in a container registry
4. At deploy time the container is pulled from the registry and run, with configuration data being provided by a runtime management environment, such as Kubernetes or OpenShift

1.1 Tutorials

The 3 tutorials in this series show you how to use Node-RED with this way of working so Node-RED can be used to develop production workloads.

- Tutorial 1 : Version Control with Node-RED
- Tutorial 2 : Package Node-RED app in a container
- Tutorial 3 : Node-RED config from environment

2. Node-RED Source Control

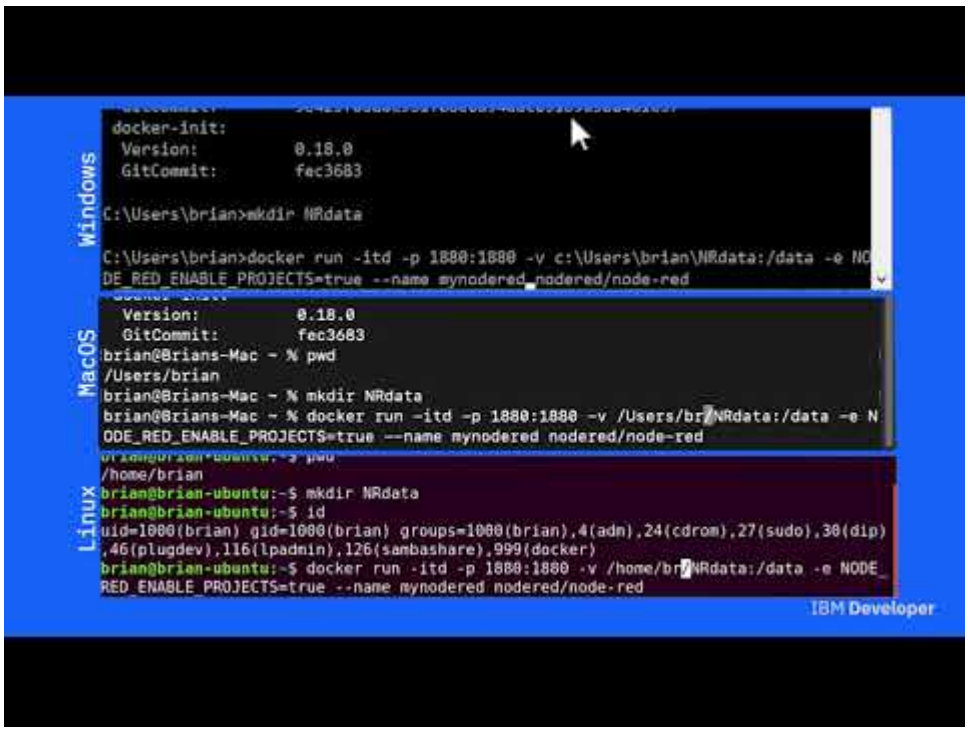
In this tutorial you will learn how to enable git integration in Node-RED. Once git integration has been turned on you will learn how to use the git integration features within the Node-RED editor.

2.1 Learning objectives

In this tutorial, you will learn how to:

- Enable Node-RED to work with git source control
- Use the git integration features of the Node-RED editor to
- clone a git repository
- commit and push changes to a git server
- pull changes from a git server
- resolve merge conflicts within the Node-RED editor

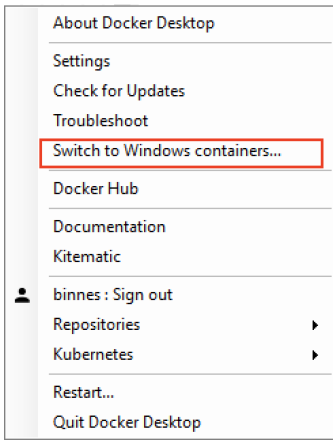
The video below shows the instructor completing the tutorial, so you can watch and follow along, or skip the video and jump to the prerequisites section.



2.2 Prerequisites

To complete this tutorial, you need:

- some experience of using Node-RED
- a laptop/workstation running an up to date version of Linux, Mac OS or Windows
- an up to date version of Docker on your laptop/workstation (version 19.03 or higher should be returned by the `docker version` command)
- Windows users need to ensure that Docker is using Linux containers. This setting is available by right clicking the Docker icon in the status section of the Windows task bar, usually at the bottom of the screen. You should see an option to switch to Windows containers. If you have an option to switch to Linux containers, you need to select it as you are currently using windows based containers.



- a github account
- git tools installed on your laptop/workstation

You will notice that a Node-RED installation is not a prerequisite. In this tutorial all development is done using a Node-RED container. Using a container ensures that all developers working on a project use a common Node-RED installation, with an '*approved*' set of Node.js packages installed in the container and nodes in the Node-RED pallet.

2.3 Estimated time

You can complete this tutorial in less than 20 minutes.

2.4 Steps

1. Codebase
2. Enable source control in Node-RED
3. Using Source control in Node-RED
4. Pulling changes and handling merge conflicts

2.4.1 Step 1. Codebase

When using Node-RED in production, you need to be able to work within a DevOps process, which relies on application source being managed by a source control system, such as git.

This is also a requirement when creating cloud native applications. The first rule for 12-factor apps is Codebase - *One codebase tracked in revision control, many deploys*

First we need to decide what is a Node-RED application code base?

A Node-RED application is defined by a flow file and an optional credentials file. However, the flow may require some additional nodes to be installed. The flow runs within a Node.js application, which is the Node-RED runtime. This runtime can be customised and configured, so to fully capture a Node-RED application code base you need to capture the :

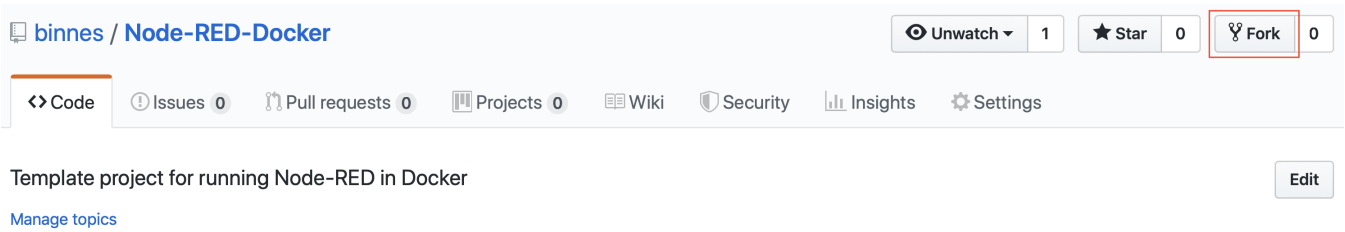
- application flow and credential file
- the package.json file, which captures all required Node.js and Node-RED nodes needed by the flow
- the Node-RED runtime source files

For this workshop a starter git project has been provided, containing a Node-RED runtime, customised to be managed by a cloud.

We are using the public github service to work through this tutorial, but Node-RED works with any standard git service. So after completing the tutorial you can choose your preferred git service or you can setup your own git server to work on premises.

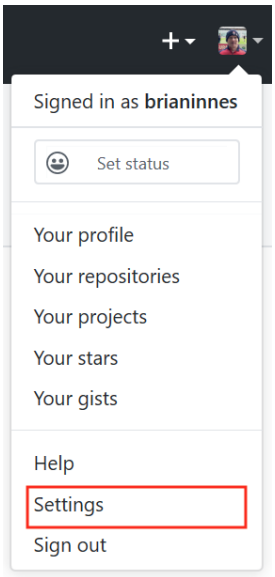
Follow the instructions to fork the starter git repository, which is contains a starter template for a new project. This will create a repository in your own github account so you can make changes to the content.

1. Open a browser an navigate to the git starter project repo
2. Make sure you are logged into your github account then press the **Fork** button so you have your own copy of the repo



3. (OPTIONAL) If you have 2-factor authentication enabled on your github account, then you need to use a Personal Access Token when using the git command line tools.

   To create a Personal Access Token: * Open the git settings



   * select **Developer settings** then **Personal access tokens** then **Generate new token**. * Give the token a use, select all scopes except `admin:enterprise` and `admin:gpg.key` scope then **Generate token**

Settings / **Developer settings**

- GitHub Apps
- OAuth Apps
- **Personal access tokens**

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

Node-RED access

What's this token for?

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

☑ **repo**	Full control of private repositories
☑ repo:status	Access commit status
☑ repo_deployment	Access deployment status
☑ public_repo	Access public repositories
☑ repo:invite	Access repository invitations
☑ **write:packages**	Upload & delete packages in github package registry
☑ **read:packages**	Download packages from github package registry
☑ **admin:org**	Full control of orgs and teams, read and write org projects
☑ write:org	Read and write org and team membership, read and write org projects
☑ read:org	Read org and team membership, read org projects
☑ **admin:public_key**	Full control of user public keys
☑ write:public_key	Write user public keys
☑ read:public_key	Read user public keys
☑ **admin:repo_hook**	Full control of repository hooks
☑ write:repo_hook	Write repository hooks
☑ read:repo_hook	Read repository hooks
☑ **admin:org_hook**	Full control of organization hooks
☑ **gist**	Create gists
☑ **notifications**	Access notifications
☑ **user**	Update all user data
☑ read:user	Read all user profile data
☑ user:email	Access user email addresses (read-only)
☑ user:follow	Follow and unfollow users
☑ **delete_repo**	Delete repositories
☑ **write:discussion**	Read and write team discussions
☑ read:discussion	Read team discussions
☐ **admin:enterprise**	Full control of enterprises
☐ manage_billing:enterprise	Read and write enterprise billing data
☐ read:enterprise	Read enterprise profile data
☐ **admin:gpg_key**	Full control of user gpg keys (Developer Preview)
☐ write:gpg_key	Write user gpg keys
☐ read:gpg_key	Read user gpg keys

Generate token Cancel

* record the token as you will need it later in the tutorial.

2.4.2 Step 2. Enable Source control in Node-RED

Node-RED has the **projects** feature, which is turned off by default. It allows Node-RED to work with a version control system to manage the Node-RED flow and associated content.

There are 2 ways of enabling projects in Node-RED:

- Update the settings.js file in the userDirectory for Node-RED (this is the .node-red folder in your home folder by default)
- Set the **NODE_RED_ENABLE_PROJECTS** environment variable before starting Node-RED

Info

In this tutorial we will run Node-RED in a Docker image, this removes the need for a local install of Node.js and node-RED, but if you have Node-RED already installed and want to enable the projects feature, then you should edit the settings.js file:

On your system edit file .node-red/settings.js in your home directory. At the bottom of the file change the projects setting to `enabled : true`

```
// Customising the editor
editorTheme: {
    projects: {
        // To enable the Projects feature, set this value to true
        enabled: true
    }
}
}
```

Follow the instructions below to start Node-RED with the project feature enabled. The commands need to be entered in a command or terminal window, running as your normal login user:

1. If Node-RED is already running on your system, stop it
2. Create a new directory called **NRdata** in your home directory to use as the Node-RED userDirectory:

   `mkdir NRdata`

   Note

   on Linux the NRdata directory needs to be writeable by user with UID 1000. If your user UID is not 1000 then make the directory writeable by everyone:

   `chmod 777 NRdata`

3. To start Node-RED use command (select your operating system. You will need to update the path to the NRdata directory):

   - **Windows**:

     `docker run -itd -p 1880:1880 -v c:\Users\YOUR-USERNAME\NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true --name mynodered nodered/node-red`

   - **Mac OS**:

     `docker run -itd -p 1880:1880 -v /Users/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true --name mynodered nodered/node-red`

   - **Linux**:

     `docker run -itd -p 1880:1880 -v /home/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true --name mynodered nodered/node-red`

   Info

   - the **-e** option is short for **--env** and sets the **NODE_RED_ENABLE_PROJECTS** environment variable
   - the **-v** option is short for **--volume** and maps your local **NRdata** directory into the container at location **/data**, which is configured as the userDirectory for Node-RED.
   - to see all the possible options for the Docker run command use `docker run --help`

2.4.3 Step 3. Using Source control in Node-RED

The git integration built into the projects feature of Node-RED will allow you to clone an existing git repository or create a new repository from within the Node-RED editor. You will also find a new **Projects** entry in the main menu, where you can change the current project, create a new project or look at the current project settings. Node-RED stores projects files in a directory called **projects** in your Node-RED user directory, this defaults to **.node-red/projects** in your home directory on your operating system. Each projects is stored in its own directory within the projects directory.

For this tutorial the Node-RED user directory is the NRdata directory you created in the previous step, so you will find project directories in **NRdata/projects**.

1. Open a browser to access your local Node-RED runtime on http://localhost:1880 and you should see the Projects wizard, as projects are enabled and no projects exist yet:

Hello! We have introduced 'projects' to Node-RED.

This is a new way for you to manage your flow files and includes version control of your flows.

To get started you can create your first project or clone an existing project from a git repository.

If you are not sure, you can skip this for now. You will still be able to create your first project from the 'Projects' menu at any time.

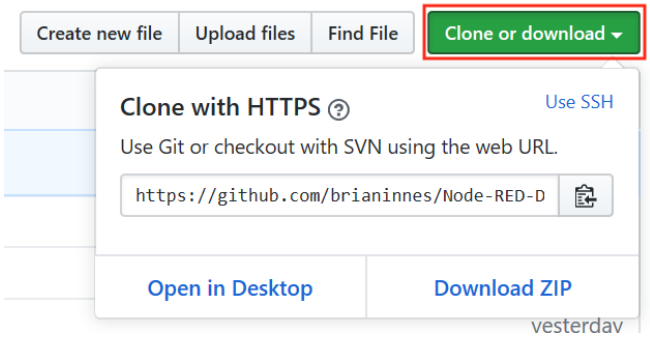
Open existing project | Not right now

None

2. Select the Clone Repository.

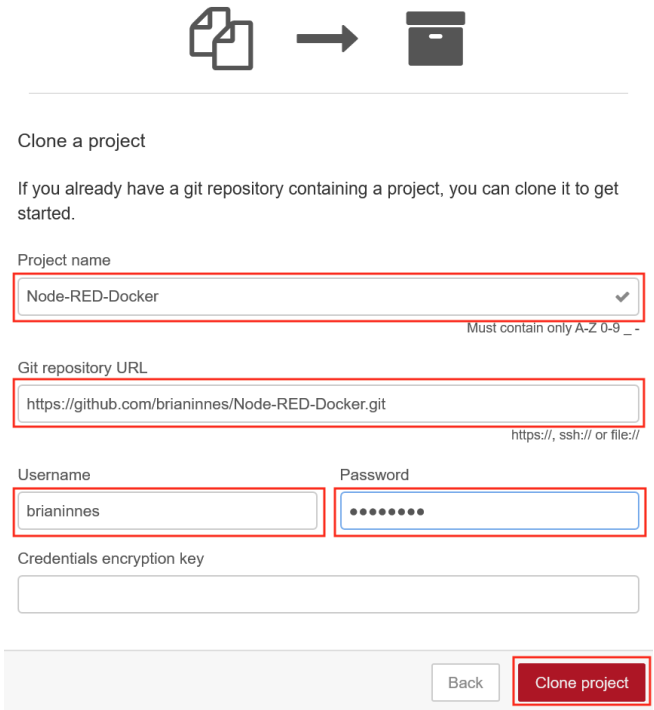
   • Enter your name and email, to be used when committing content.

   • copy the Git repo URL from **your** git repo github page (the URL should contain your github user name):



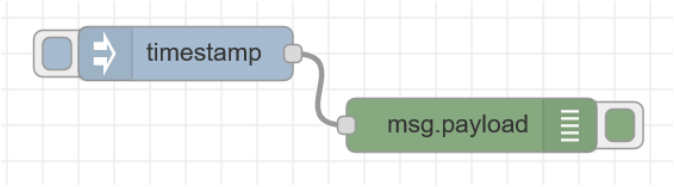
   • Back in the Node-RED editor window, paste the github URL into the **Git repository URL** field.

   • Enter your git credentials for the Username and Password fields (use the Personal Access Token as the password if you have one set on your github account - your github account password will not work if a personal access token is defined) then press the **Clone project** button when all the details have been completed - leave the Credentials encryption key field blank

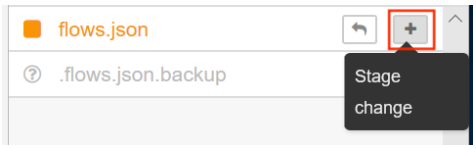


> **Info**
>
> Leaving the **credentials encryption key** field blank means that any credentials entered in any node configuration will be checked into git unencrypted. For this tutorial this isn't an issue as we want to go look at the credentials. For your own projects you may want to enter an encryption key, unless you plan to provide all credentials at run time and want to inspect the credentials file to verify there are no captured credentials.

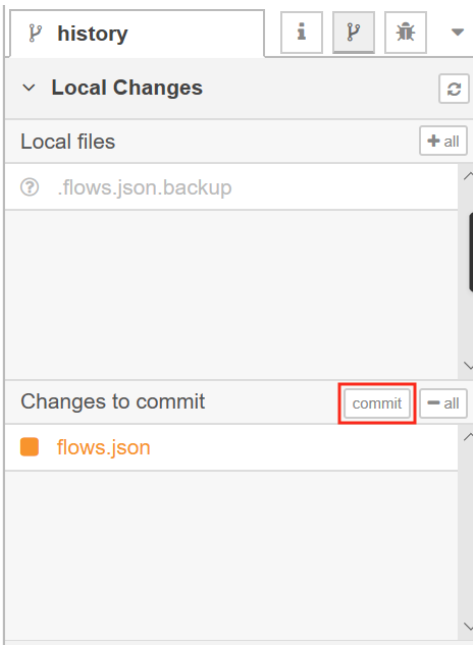
3. Drag an inject and debug node onto the sheet and connect them to create a basic flow then deploy the changes



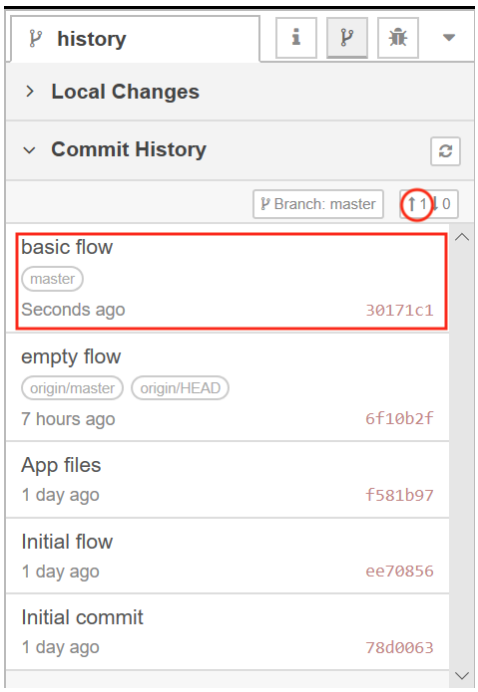
4. Switch to the project history section in the side panel, where you can see the flows.json file has uncommitted local changes. Move your mouse over the entry and press the + button to stage the change



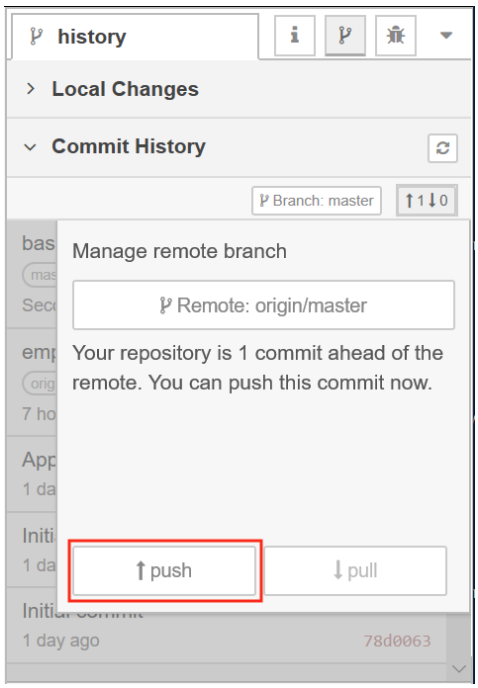
5. The flows.json file is now ready to be committed. Press the commit button then enter a commit message **basic flow** to commit the flow change



6. Switch to the **Commit History** section of the project side panel. Here you can see the last commit and also that the local git branch is 1 commit ahead of the remote master. Click the up arrow to open the Manage remote branch panel



7. Press the **Push** button to send the commit to the remote branch



> **Note**
>
> There is an issue when running on Windows 10 - the git push is not always passing the git credentials correctly. If this is the case you can issue a `git push` command from a command window. Change to the active git project directory : \[*user home directory*]\NRdata\projects\[*repo name*] then issue the git push command.

2.4.4 Step 4. Pulling changes and handling merge conflicts

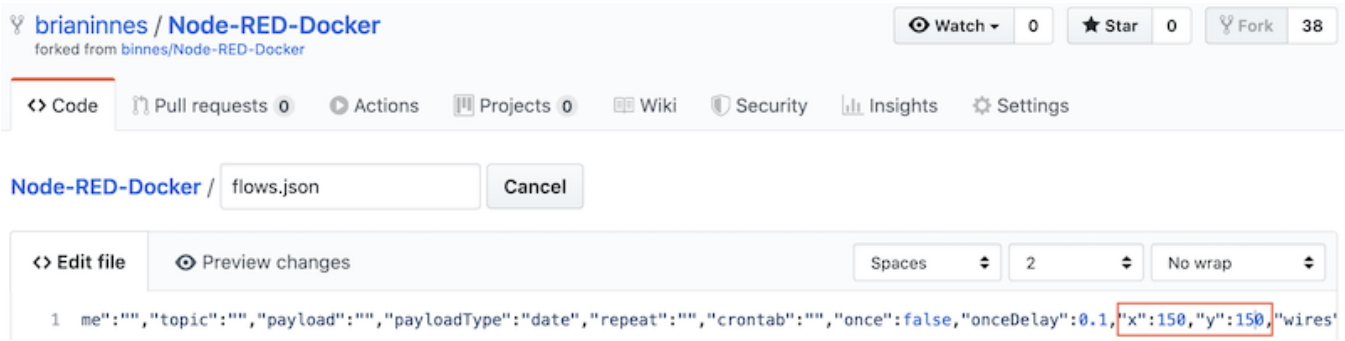
You can also pull changes from the repo, so if a team is working on the same Node-RED application they can push their individual changes and the other team members can pull the changes.

It is recommended that team members use different tabs in the Node-RED editor when collaborating to avoid merge conflicts, but if a merge conflict does occur, then it can be resolved within the Node-RED editor.

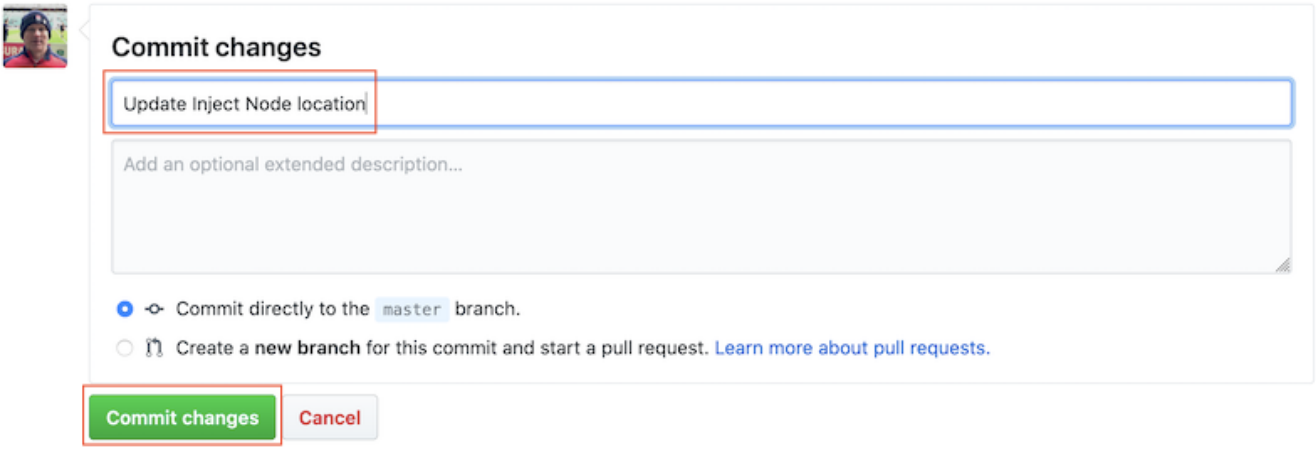
Follow the instructions below to cause a merge conflict then resolve it.

1. In the GitHub web UI open your forked project then open the flows.json page
2. Select the pencil icon to start editing the file

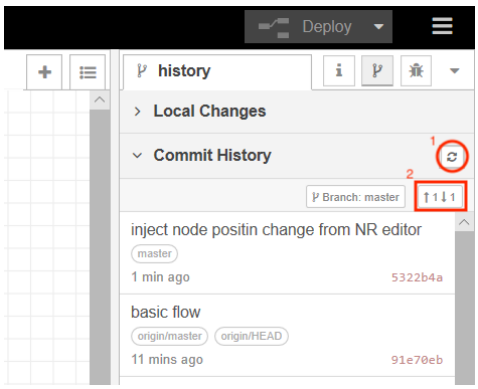
3. Find the x and y coordinates of the inject node and modify them, add or remove 10 to each of the values for x and y



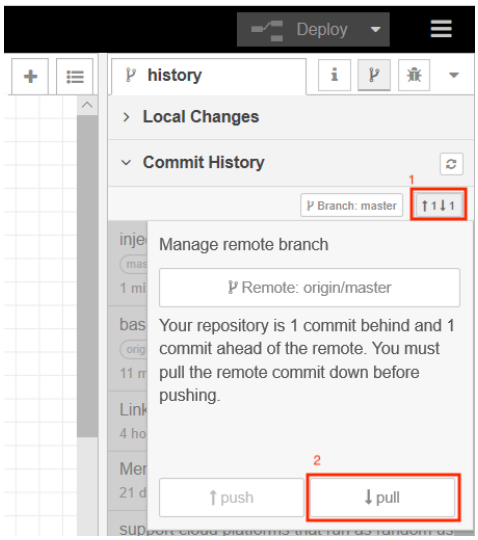
4. Press the **Commit changes** to save the change and commit the change to the master branch of the GitHub repo.



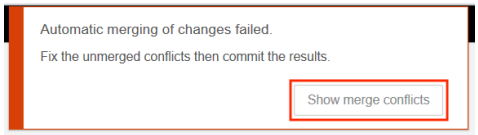
5. Back in the Node-RED editor move the inject node to a new location, then press the **Deploy** button to make the change live.
6. Open the git section of the side panel then stage and commit the update to the flow.
7. Open up the Commit History section and refresh the panel. You will now see 1 change to push and 1 change to pull.



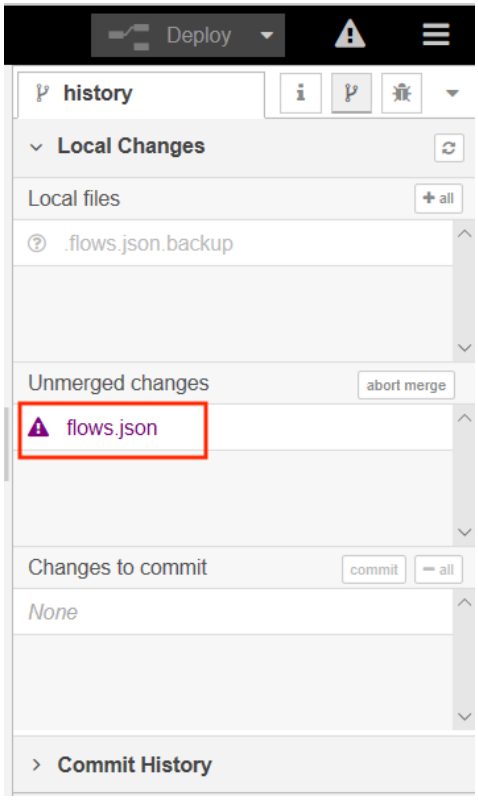
8. Clicking to manage the remote branch you will see the pull button enabled, select the pull button to bring in the change you made directly in GitHub



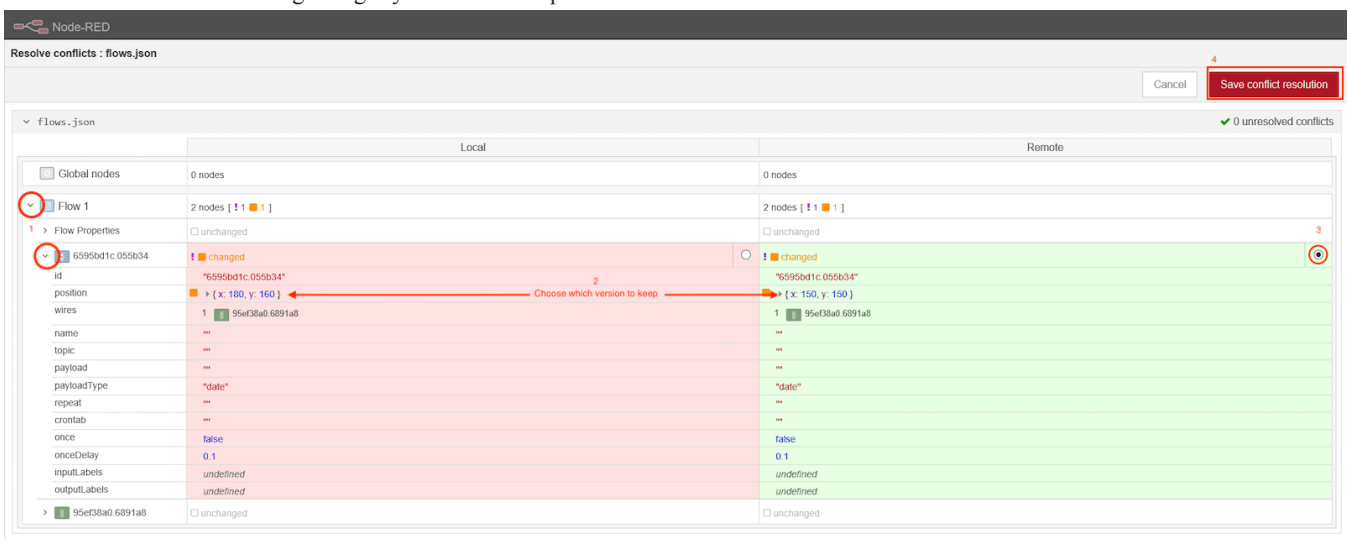
9. You should get a notification that automatic merging failed - which is expected as the GitHub change and your local change in the Node-RED editor made different changes to the same node. Select the **Show merge conflicts** button in the notification window



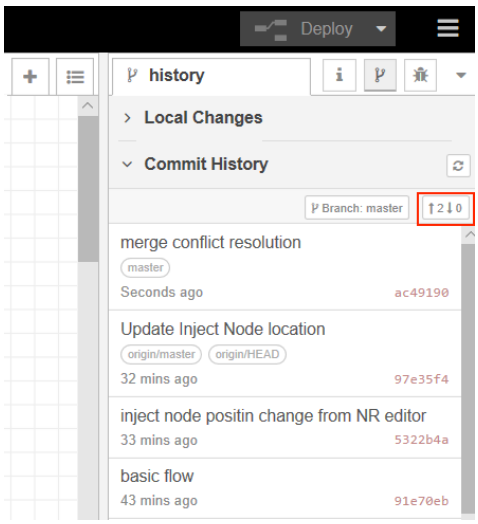
10. There is now a new section in the git panel - **Unmerged changes**. Click the flow.json file to open the resolve conflicts window, highlighting where the conflicts are.



11. Expand the twisties until you can see the conflicting positions for the inject node. You will also notice a radio button allowing you to select which version of the conflicting changes you want to accept.



12. Select one of the versions by clicking the radio button. The version which will be saved is highlighted in green and the version that will be overwritten is highlighted in red. When all conflicts have been resolved click the **Save conflict resolution** button.
13. There is a new change to commit - this is the merge resolution commit. Commit the change then navigate to the Commit History section of the Git side panel. Here you will see there are no incoming changes, but there are 2 outgoing changes. The original committed change and the merge resolution change. Select **Manage remote branch** then push the changes.



2.5 Useful Docker command

Here are a few useful Docker commands:

- `docker ps -a` : list all containers (running and stopped)
- `docker stop mynodered` : will stop the container instance named **mynodered**, but leave the container resources intact
- `docker start mynodered` : will start a stopped container instance named **mynodered**
- `docker rm mynodered` : will remove all resources for container instance named **mynodered**. The container instance must be stopped before it can be removed

You can remove the **mynodered** container instance without loosing any data, as the docker run command mapped the **NRdata** directory into the container, so all Node-RED data has been persisted in that directory, outside the container.

2.6 Summary

In this tutorial you:

- Enabled the projects feature in the Node-RED editor to provide integration with Git version control systems
- Cloned a GitHub project from within the Node-RED editor
- Committed and pushed changes to GitHub
- pulled changes from GitHub and resolved a merge conflict within the Node-RED editor

You can now work with Node-RED in a team environment synchronising via a Git repository, ensuring all changes to your Node-RED applications are version controlled.

Move onto the next tutorial, where you will learn how to package your application into a container.

3. Packaging Node-RED apps in containers

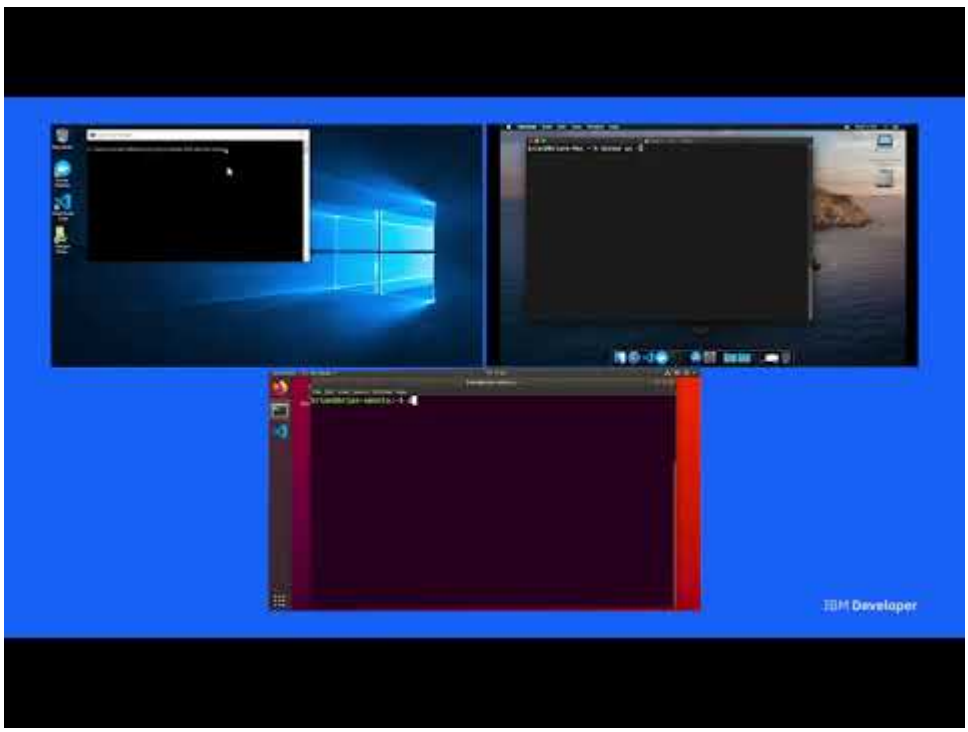
In this tutorial you will learn how to package a Node-RED application into a container and customise the runtime for production running rather than application creation

3.1 Learning objectives

In this tutorial, you will learn how to:

- Identify the dependencies for a Node-RED application
- Customise the Node-RED runtime for production use
- Build a multi-architecture container for a Node-RED application and push it to a container registry
- Run a containerised Node-RED application

The video below shows the instructor completing the tutorial, so you can watch and follow along, or skip the video and jump to the prerequisites section.



3.2 Prerequisites

To complete this tutorial, you need:

- some experience of using Node-RED
- a laptop/workstation running an up to date version of Linux, Mac OS or Windows
- an up to date version of Docker on your laptop/workstation (version 19.03 or higher should be returned by the `docker version` command)
- An active Docker ID to be able to sign into dockerhub
- a github account
- git tools installed on your laptop/workstation
- Completed tutorial **Version Control with Node-RED** and have the forked template repository in your github account, which is also cloned within a Node-RED project on your laptop/workstation

3.3 Estimated time

You can complete this tutorial in less than 20 minutes.

3.4 Steps

1. Dependencies
2. Extend application
3. Build the application

3.4.1 Step 1. Dependencies

Containers are becoming the standard way to package, distribute and deploy applications for modern cloud based environments. Increasingly containers are also being used to distribute and manage workloads in edge of network scenarios.

X86_64 is the predominant CPU architecture in use today in public cloud, but there are also Open Power and S390 systems being used in infrastructures running many large businesses. At the edge there are also ARM32 and ARM64 systems, so in this tutorial you will build a multi-architecture container, allowing your containers to run on multiple different architectures.

The 12-factor app dependency rule is to *Explicitly declare and isolate dependencies*

For a Node-RED application all dependencies are specified in the package.json file. When adding additional nodes to the Node-RED pallet, ensure they are added to the package.json.

However, there are still '*hidden*' dependencies that can creep into a project when a package has some native dependencies that need to be installed in the host system running the application.

To get round this the starter project has a Dockerfile which will build the application from the source code and package it into a container. The Dockerfile captures all *hidden* dependencies.

The provided Dockerfile in the starter project initially creates a build container to build the required software, then creates an applications image, copying built content from the build container. This way a fully defined build environment is created and used, but the build tooling is not part of the production container image.

The runtime files in the starter project have been modified to allow the application to be better managed in the cloud. Additional endpoints (*/live, /ready and /health*) have been added to allow a cloud environment to verify the state of the running container. The editor has also been moved to the **/admin** endpoint. In a production environment the editor should be disabled, but for this tutorial it has been left active to allow you to examine the Node-RED runtime running in the container.

In this tutorial the new **buildx** Docker feature is used to make it easier to create and push multi-architecture containers.

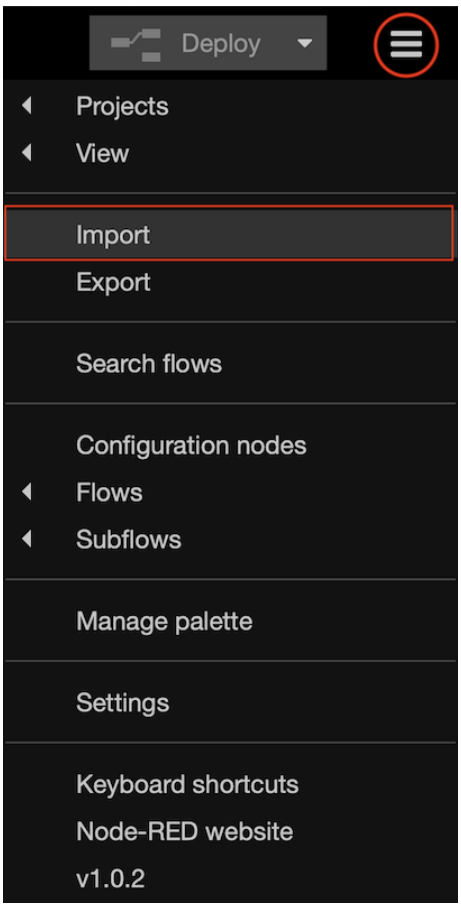
3.4.2 Step 2. Extend the application

Before building the app we will add another few nodes to add a Web endpoint, so we can test an app when we have Node-RED deployed in Docker.

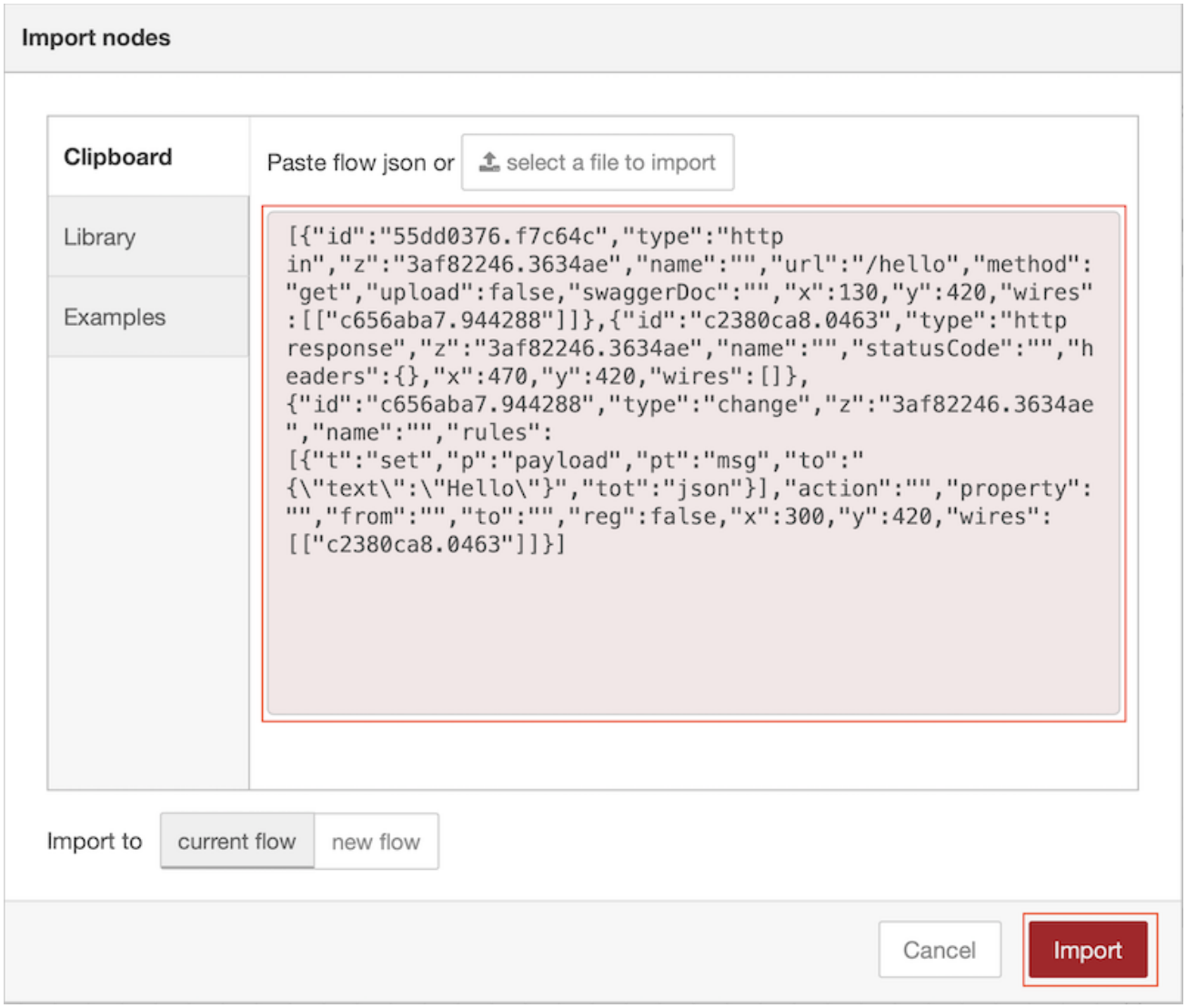
1. Start Node-RED, if not already running:
    - run `docker ps -a` to see what is running
    - if **mynodered** container instance exists, but is not in state **up**, then run command `docker start mynodered`
    - if **mynodered** container instance does not exist then run the appropriate command (replacing *YOUR-USERNAME* with your own username):
    - **Windows**:
      `docker run -itd -p 1880:1880 -v c:\Users\YOUR-USERNAME\NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true --name mynodered nodered/node-red`
    - **Mac OS**:
      `docker run -itd -p 1880:1880 -v /Users/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true --name mynodered nodered/node-red`
    - **Linux**:
      `docker run -itd -p 1880:1880 -v /home/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true --name mynodered nodered/node-red`

2. Import the following JSON to add the **/hello** endpoint:

• to import the flow select the main menu (), then the import option from the menu


Deploy
Projects
View
Import
Export
Search flows
Configuration nodes
Flows
Subflows
Manage palette
Settings
Keyboard shortcuts
Node-RED website
v1.0.2

• copy and paste the JSON below into the Import nodes window, then press the **Import** button to import the nodes




```
[{"id":"55dd0376.f7c64c","type":"http in","z":"3af82246.3634ae","name":"","url":"/hello","method":"get","upload":false,"swaggerDoc":"","x":130,"y":420,"wires":
[["c656aba7.944288"]]},{"id":"c2380ca8.0463","type":"http response","z":"3af82246.3634ae","name":"","statusCode":"","headers":{},"x":470,"y":420,"wires":[]},
{"id":"c656aba7.944288","type":"change","z":"3af82246.3634ae","name":"","rules":[{"t":"set","p":"payload","pt":"msg","to":"{\"text\":
\"Hello\"}","tot":"json"}],"action":"","property":"","from":"","to":"","reg":false,"x":300,"y":420,"wires":[["c2380ca8.0463"]]}]
```


• press the **Deploy** button to make the new nodes live (you can now access the new endpoint running on your local Node-RED instance http://localhost:1880/hello.

> **Note**
>
> This web endpoint generates a JSON response. Most browsers can display JSON content, but not all can without having a plugin installed. If you get prompted to install a plugin you can choose to install one, or just take the request as validating that the endpoint worked*

3. Commit and push the change to git
    - switch to the git side panel
    - stage the change to the flows.json
    - commit the change
    - switch to the Commit History section of the side panel
    - click the up arrow to open the Manage remote branch popup
    - push the change to the server

3.4.3 Step 3. Configure the builder and build the application

In this tutorial we will use the new **buildx** feature of Docker. At the time of writing this content it is an experimental feature in Docker, so experimental features need to be enabled in Docker to get access to buildx:

1. Enable buildx in Docker:
   - Linux
   - Environment variable **DOCKER_CLI_EXPERIMENTAL** should be set to **enabled**

     This can be done on the command line, or added to your .profile or active config file sourced when a new login shell is launched:

     ```
     export DOCKER_CLI_EXPERIMENTAL=enabled
     ```

   - To build multi-architecture images on Linux, architecture emulation needs to be added to Linux. This can be done by running the following command:

     ```
     docker run --rm --privileged docker/binfmt:66f9012c56a8316f9244ffd7622d7c21c1f6f28d
     ```

   - MacOS and Windows
   - Start Docker if it is not running
   - Click the Docker icon (usually in bottom notification popup on Windows, top menu bar on MacOS) and select **settings** or **Preferences** then the **Command Line** section
   - Enable Experimental features
   - Open a command or terminal window then navigate to the project directory:
   - navigate to your home directory
   - navigate to the **NRdata/projects/Node-RED-Docker** subdirectory within your home directory. This directory should contain the Dockerfile.
   - Before you can build a container you need to create a new builder. Enter the command:

```
docker buildx create --name NRbuilder --use
```

2. Inspect the builder with command :

   ```
   docker buildx inspect --bootstrap
   ```

   which will also start the builder if it is not running. The output of this command will show the target architectures supported by the builder. 5. You can check you have a builder running using the **ls** command, which also outputs the list of supported architectures :

   ```
   docker buildx ls
   ```

3. Now the builder is up and running you can build a multi-arch container and push it to your dockerhub account. First ensure you are logged into dockerhub :

   ```
   docker login
   ```

4. Build and push the image :

   ```
   docker buildx build --platform linux/amd64,linux/arm64,linux/arm/v7 -t YOUR-DOCKER-USERNAME/node-red-docker-sample --push .
   ```

   replace **YOUR-DOCKER-USERNAME** with your docker username. Here you see we are asking to build an image for 3 different architectures. AMD/Intel 64 bit, ARM 64bit and ARM 32bit v7 (Raspberry Pi 3/4). You can change to list of architectures to build as needed, e.g. adding additional architectures, such as **linux/s390x** to add support for IBM Z systems or **linux/ppc64le** for IBM POWER systems.

> **Warning**
>
> the more architectures you select to build, the longer the build takes. The list of architectures your build environment supports is provided in the output to the `docker buildx ls` command.

- the **-t** option is short for **--tag** which applies a tag to the container image in the registry
- to see all the options available when building an image use command `docker buildx build --help`

5. Inspect the image using command (replace **YOUR-DOCKER-USERNAME** with your docker username):

```
docker buildx imagetools inspect docker.io/YOUR-DOCKER-USERNAME/node-red-docker-sample:latest
```

6. Stop your local Node-RED

   (we want to test the new container and will use the same Node-RED port of 1880, so can't have 2 applications listening on the same port):

   `docker stop mynodered`

   in a command line window, start your new container using command :

   ```
   docker run -dit -p 1880:1880 --name dockerNR YOUR-DOCKER-USERNAME/node-red-docker-sample:latest
   ```

7. Test your container.
   - You will not be able to launch at the Editor on the base URL, as this has been modified in the sample project **settings.js** file. The editor can be launched at **/admin**. In a production Node-RED container you should not be able to alter the application, so the editor needs to be disabled. This can be achieved by setting the **httpAdminRoot** property in the **settings.js** file to **false**. Details of the Node-RED configuration options can be found in the Node-RED documentation.
   - You should be able to access the /hello endpoint
8. If you have a Raspberry Pi or other ARM 32-bit or ARM 64-bit system you can also test that the ARM containers also work.

3.5 Summary

In this tutorial you:

- Enabled experimental features in Docker to access the buildx command
- Created a new builder instance
- Created a multi-architecture set of containers and pushed them to dockerhub
- Inspected the created images
- Ran the newly created container on your local machine and optionally on a system with a different CPU architecture

Now you can create a Docker image containing your Node-RED application, but to make the container suitable for running in a cloud environment there is a further consideration needed to allow the cloud environment to provide configuration at runtime to the container, which is the subject of the next tutorial.

4. Node-RED configuration from environment

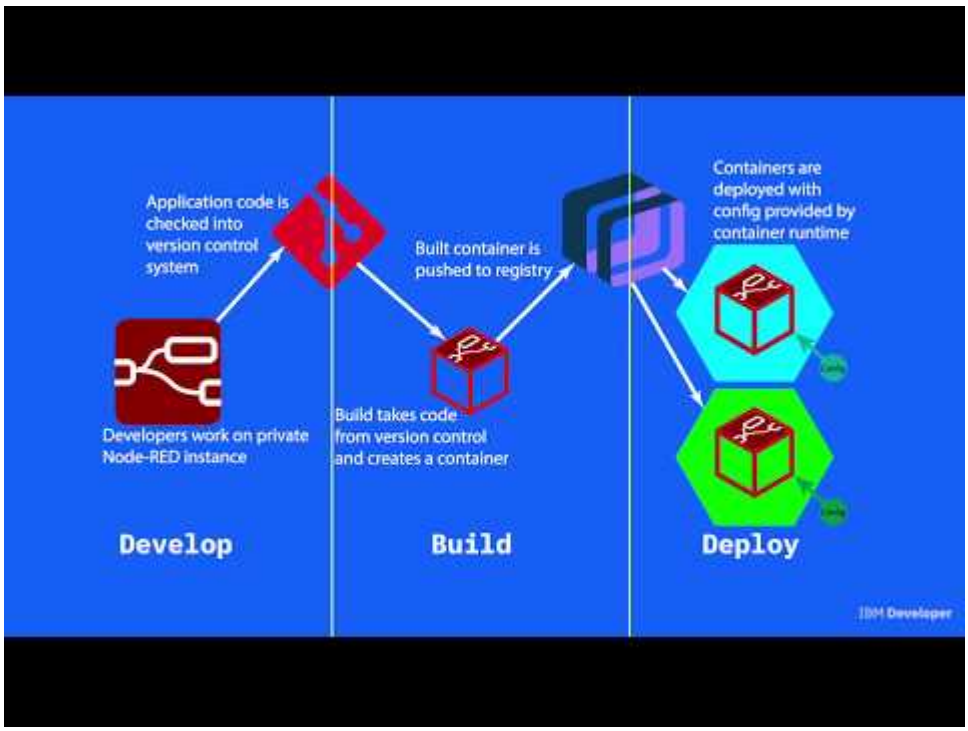
In this tutorial you will learn how to create Node-RED applications able to receive their configuration from the environment at runtime

4.1 Learning objectives

In this tutorial, you will learn how to:

- get configuration for nodes from the environment
- access environment variables within a flow
- set environment variables when running a Docker container
- enable containers to communicate using a Docker user-defined bridge

The video below shows the instructor completing the tutorial, so you can watch and follow along, or skip the video and jump to the prerequisites section.



4.2 Prerequisites

To complete this tutorial, you need:

- some experience of using Node-RED
- a laptop/workstation running an up to date version of Linux, Mac OS or Windows
- an up to date version of Docker on your laptop/workstation (version 19.03 or higher should be returned by the `docker version` command)
- a github account
- git tools installed on your laptop/workstation
- Completed tutorial **Package a Node-RED application in a container** and have the forked template repository in your github account, which is also cloned within a Node-RED project on your laptop/workstation

4.3 Estimated time

You can complete this tutorial in less than 20 minutes.

4.4 Steps

1. Environment Variables
2. Run a local broker
3. MQTT node config
4. Removing static config from nodes
5. Testing the environment variable substitution
6. Updating Docker container in dockerhub

Having configuration embedded in a container means that container is restricted to a single environment. If a container application needs to connect to external services, such as a database or messaging service then allowing the configuration details for the external services to be provided at runtime makes the container much more useful, as it doesn't need to be rebuilt every time configuration changes.

The 12-factor app Config rule requires that config is stored in the environment.

Cloud runtime environments pass configuration to applications through a number of different mechanisms. Setting environment variables is a common approach.

In Node-RED static configuration is usually found within the configuration properties of the nodes. In the previous tutorial, when you enabled projects within the Node-RED editor you disabled configuration encryption, so it is easy to see the configuration, including the password configuration in the flows_cred.json file in the project directory.

4.4.1 Step 1. Environment Variables

A number of nodes have the ability to access environment variables from their configuration, e.g. Inject node, Change node and Switch node.

It is also possible get a node configuration property to be replaced by an environment variable when the flow is loaded by using syntax **${ENV_VAR}** in any string based configuration property.

To set an environment variable named **WWW** in a command window:

- Linux and MacOS : `export WWW=123`
- Windows : `set WWW=123`

This sets the environment variable only for the current terminal or command window session. There are ways on the different operating systems to set environment variables, so they are always set.

To see all environment variable that are set, simply enter command `set` on Linux, MacOS and Windows and you will see all the environment variables that are set.

Environment variables from the host are not automatically passed into a docker container. You need to use the -e command line option to set an environment variable as part of the command line. You can either provide a value for the environment variable on the command line or pass in the value from the local environment:

- `docker run -e WWW=987 ...` : will set the value 987 for environment variable **WWW** inside the container instance
- `docker run -e WWW ...` : will set the value for **WWW** from the local environment when the docker run command is executed. If the **WWW** was set as above then the value in the container instance would be 123. If there is no local environment variable set for **WWW** then the variable will not be set in the container

You can also use the **--env-file** option to read environment variables from a file:

`docker run --env-file env.list ...`

with file **env.list** containing

```
# This line is a comment line
VAR1=abc
WWW
```

the **VAR1** environment variable would be set to *abc* and **WWW** would be set to the value of the local environment variable **WWW** when the docker command was run.

1. Stop any running Node-RED instances, then start Node-RED setting the WWW environment variable (If you have the Docker container from the previous tutorial still running then you need to stop that):

   - `docker stop dockerNR`
   - `docker rm dockerNR`
   - `docker stop mynodered`
   - `docker rm mynodered`
   - choose the appropriate command for your operating system (replacing *YOUR-USERNAME* with your own username):
   - **Windows**:

     ```
     docker run -itd -p 1880:1880 -v c:\Users\YOUR-USERNAME\NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -e WWW=123 --name mynodered nodered/node-red
     ```

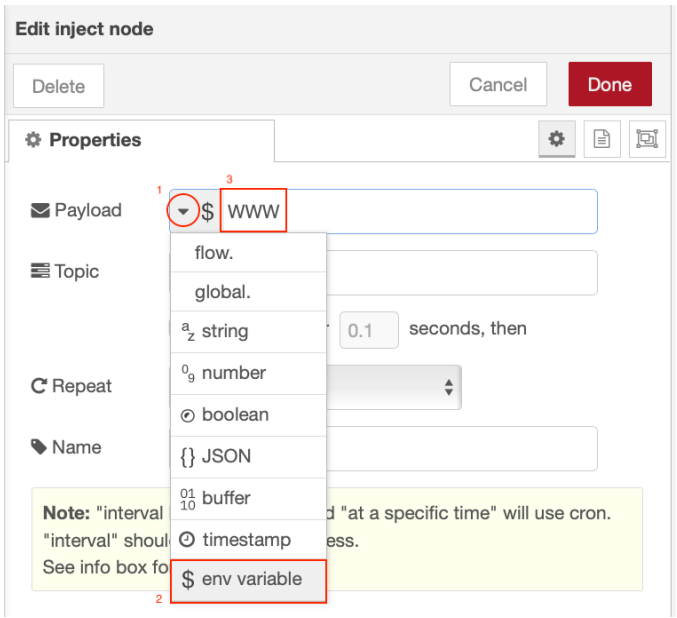
   - **Mac OS**:

     ```
     docker run -itd -p 1880:1880 -v /Users/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -e WWW=123 --name mynodered nodered/node-red
     ```

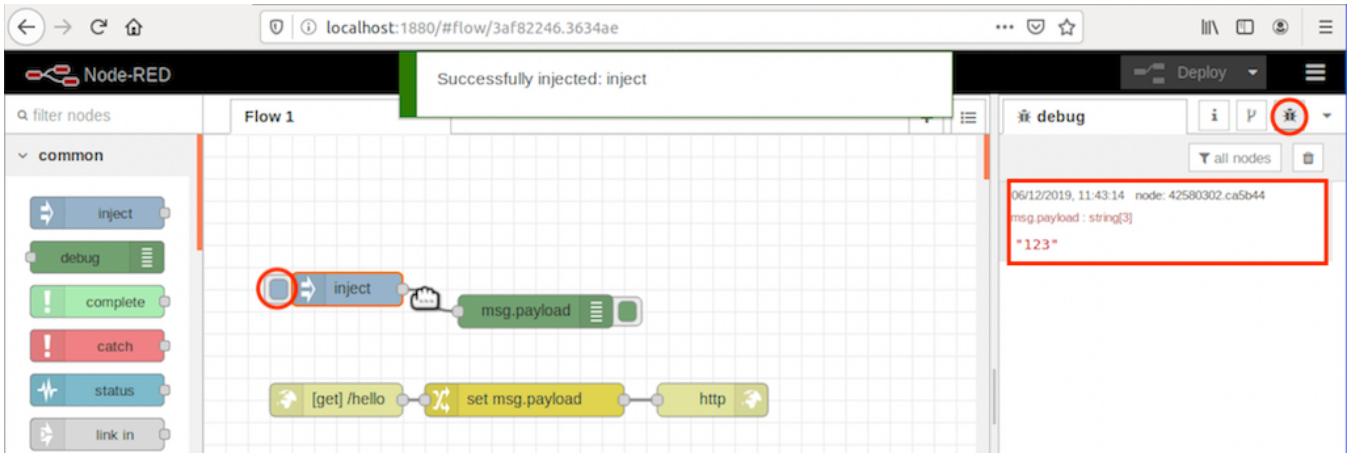
   - **Linux**:

     ```
     docker run -itd -p 1880:1880 -v /home/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -e WWW=123 --name mynodered nodered/node-red
     ```

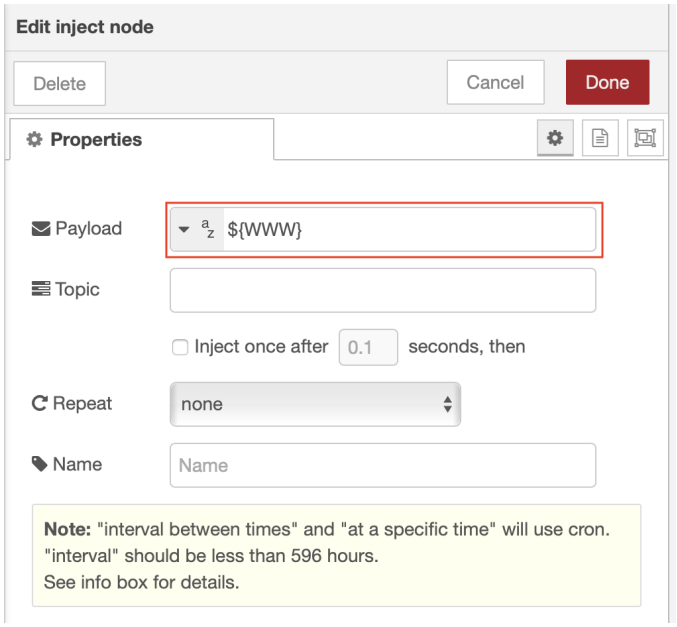
2. Modify the inject node you added in the **Node-RED with source control** tutorial to set the payload to the value of the **WWW** environment variable



   Deploy the change then test the outcome by pressing the button on the Inject node to send a message. Switch to the debug tab to see the outcome



3. Modify the inject node configuration to inject a string and set the string value to **${WWW}** which produces the same result as using the env variable setting:



Test the outcome by deploying the change, then press the button on the inject node - the output should be the same as the previous output, showing environment variable substitution.

4.4.2 Step 2. Run a local broker

1. You need a clone a git repository to your local system so you can access the SSL certificates we will use for the rest of this tutorial:

   `git clone https://github.com/binnes/moreNodeRedWorkshop.git`

   or if using ssh keys with GitHub:

   `git clone git@github.com:binnes/moreNodeRedWorkshop.git`

2. Add a docker network bridge to allow the Node-RED container to find the MQTT broker container. A user-defined network bridge allows containers connected to the same bridge to communicate, without exporting ports and also provides automatic name resolution using the --name parameter provided at container start:

   `docker network create NRbridge`

3. Start the MQTT Mosquitto container:

   `docker run -itd -p 8883:8883 -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mqttBroker eclipse-mosquitto`

   replace the `<full path to where moreNodeREDWorkshop repo cloned>` with the **fully qualified path** of the directory containing the git repository.

   e.g. On windows, if I cloned the repository into my home directory **c:\Users\brian** then the command would be:

   `docker run -itd -p 8883:8883 -v c:\Users\brian\moreNodeRedWorkshop\en\part5\broker:/mosquitto --network NRbridge --name mqttBroker eclipse-mosquitto`

   On Mac or Linux, if I cloned the repository into home directory **/Users/brian** then the command would be:

   `docker run -itd -p 8883:8883 -v /Users/brian/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mqttBroker eclipse-mosquitto`

   > **Info**
   >
   > The -p option is passed here to export port 8883. This exposes the MQTT broker port 8883 as if it were installed directly onto your laptop or workstation. If you only want the services of the MQTT broker to be available to other containers connected to the NRbridge network, then you can omit the -p 8883:8883 from the command line.

4. (OPTIONAL) If you want to create additional broker users in the container. There is already a default user created, the username is **mosquitto** with password **passw0rd**:

   `docker exec mqttBroker mosquitto_passwd -b /mosquitto/config/passwd username userpassword`

   replacing *username* and *userpassword* as required

5. If you want to watch the mosquitto logs you can with command:

   `docker logs -f mqttBroker`

   Press and hold the Control key, then press C (Ctrl-C) to exit following the mqtt broker logs

4.4.3 Step 3. MQTT node config

In this section you will add some additional nodes to Node-RED, which connect to your local MQTT broker.

1. Restart the Node-RED service on your system. To connect to the broker the Node-RED container need access to the broker certificates, so we will map the same volume as we did when starting the broker, as the certificates are in a cert sub-directory. We also need to add the --network option to put the Node-RED container instance on the same Docker network bridge as the MQTT broker container instance.

   `docker stop mynodered`

   `docker rm mynodered`

   Now run the appropriate command for your operating system (replacing *YOUR-USERNAME* with your own username): * **Windows**:

   ```
   `docker run -itd -p 1880:1880 -v c:\Users\YOUR-USERNAME\NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -v <full path to where moreNodeREDWorkshop repo cloned>\moreNodeRedWorkshop\en\part5\broker:/mosquitto --network NRbridge --name mynodered nodered/node-red`
   ```

   - **Mac OS**:

     `docker run -itd -p 1880:1880 -v /Users/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mynodered nodered/node-red`

   - **Linux**:

     `docker run -itd -p 1880:1880 -v /home/YOUR-USERNAME/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mynodered nodered/node-red`

   e.g. on mac, logged in as user brian, the command might look like:

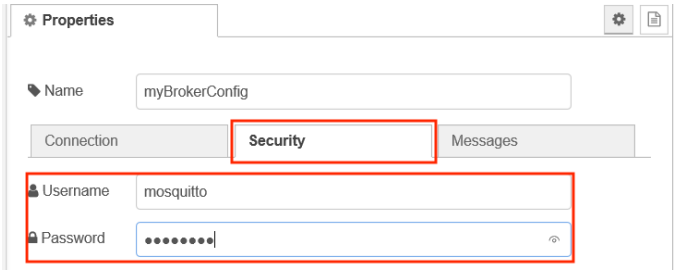
   `docker run -itd -p 1880:1880 -v /Users/brian/NRdata:/data -e NODE_RED_ENABLE_PROJECTS=true -v /Users/brian/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mynodered nodered/node-red`

2. Import the following flow (using the same technique used in the previous tutorial - **menu** -> **import**):

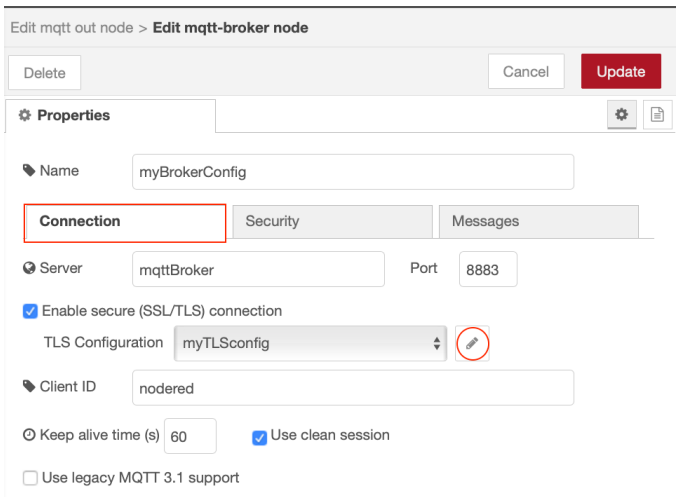
   ```
   [{"id":"8381a5c3.3cbf9","type":"mqtt in","z":"3af82246.3634ae","name":"","topic":"#","qos":"2","datatype":"json","broker":"d2a17e7.00b668","x":90,"y":660,"wires":
   [["ebab6856.8901c"]]},
   {"id":"ebab6856.8901c","type":"debug","z":"3af82246.3634ae","name":"","active":true,"tosidebar":true,"console":true,"tostatus":false,"complete":"true","targetType":"f
   410,"y":660,"wires":[]},
   {"id":"92d9d4a9.e5cf4","type":"inject","z":"3af82246.3634ae","name":"","topic":"","payload":"","payloadType":"str","repeat":"10","crontab":"","once":false,"onceDelay"
   0.1,"x":110,"y":540,"wires":[["fb7b0c12.eb2e48"]]},{"id":"fb7b0c12.eb2e48","type":"change","z":"3af82246.3634ae","name":"","rules":
   [{"t":"set","p":"payload","pt":"msg","to":"{ \"time\" : $fromMillis($toMillis($now()),'[H]:[m]:
   [s]') }","tot":"jsonata"}],"action":"","property":"","from":"","to":"","reg":false,"x":260,"y":560,"wires":[["d356084b.b81818"]]},
   {"id":"d356084b.b81818","type":"mqtt out","z":"3af82246.3634ae","name":"","topic":"time","qos":"","retain":"","broker":"d2a17e7.00b668","x":410,"y":580,"wires":
   [[]],{"id":"d2a17e7.00b668","type":"mqtt-
   broker","z":"","name":"myBrokerConfig","broker":"mqttBroker","port":"8883","tls":"9ec473ef.e0678","clientid":"nodered","usetls":true,"compatmode":false,"keepalive":"6
   {"id":"9ec473ef.e0678","type":"tls-config","z":"","name":"myTLSconfig","cert":"","key":"","ca":"/mosquitto/certs/
   mqtt_ca.crt","certname":"","keyname":"","caname":"","servername":"mqttBroker","verifyservercert":true}]
   ```

3. Open up the configuration of either of the mqtt nodes and select the edit icon next to the MQTT server config

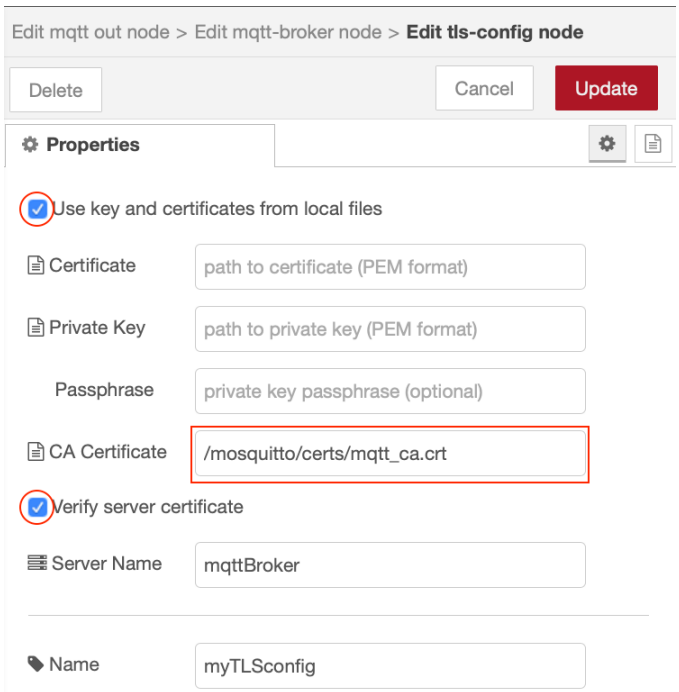
4. Switch to the **Security** tab and enter the MQTT broker credentials. The default credentials are **mosquitto** / **passw0rd**



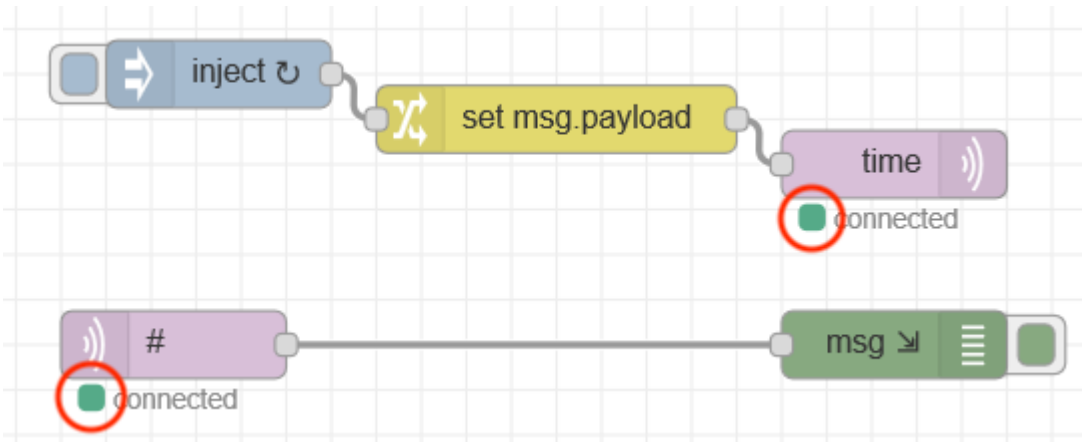
5. Switch to the **Connection** tab and open the TLS Configuration



6. The root certificate information should be already populated. The certificate is read from the imported volume Docker mapped to the /mosquitto path



7. Press the update and Done buttons to save the configuration, then Deploy the flow. You should see the MQTT nodes connected to your mqtt broker



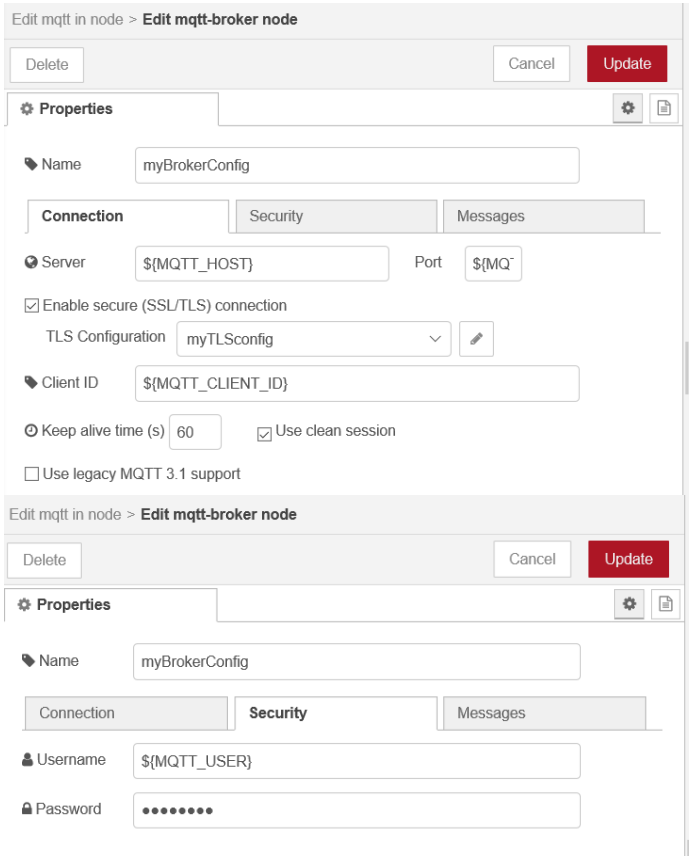
The sample application publishes the time to the configured MQTT broker every 10 seconds. The second MQTT node subscribes to all topics (using # wildcard), so will receive all messages published by the first node. It simply writes out the received message to the debug panel and also the system console (which makes the messages visible in the logs, which can be accessed using `docker logs -f mynodered` command)

4.4.4 Step 4. Removing static config from nodes

To prevent config being captured in a flow you can replace all configuration of nodes by environment variables, so at run time the environment can provide the configuration to a flow, rather than the configuration being trapped in the flow. This works for all string based values (including passwords).

1. Open up the server config and make the following changes on the Connection tab:
    - set the Server to ${MQTT_HOST}
    - set the Port to ${MQTT_PORT}
    - set the Client ID to ${MQTT_CLIENT_ID}
2. Switch to the Security tab:
    - set the Username to ${MQTT_USER}
    - set the Password to ${MQTT_PWD} (you won't be able to see this, as the password field hides the content)
3. Switch back to the Connection tab and open the TLS config editor
    - set the CA Certificate to ${MQTT_CA_CERT}
    - set the Server name to ${MQTT_HOST}
4. Press Update and Done to close the config panels then Deploy the flow.

This is what the config should now look like:



and the resultant flow file segment :

```
{
    "id": "7a44476d.a179c8",
    "type": "mqtt-broker",
    "z": "",
    "name": "myBrokerConfig",
    "broker": "${MQTT_HOST}",
    "port": "${MQTT_PORT}",
    "tls": "850bd469.ceb218",
    "clientid": "${MQTT_CLIENT_ID}",
    "usetls": true,
    "compatmode": false,
    "keepalive": "60",
    "cleansession": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthRetain": "false",
    "birthPayload": "",
    "closeTopic": "",
    "closeQos": "0",
    "closeRetain": "false",
    "closePayload": "",
    "willTopic": "",
    "willQos": "0",
    "willRetain": "false",
    "willPayload": ""
},
{
    "id": "850bd469.ceb218",
    "type": "tls-config",
    "z": "",
    "name": "myTLSconfig",
    "cert": "",
    "key": "",
    "ca": "${MQTT_CA_CERT}",
    "certname": "",
    "keyname": "",
    "caname": "",
    "servername": "${MQTT_HOST}",
    "verifyservercert": true
},
```

and flow credentials file:

```
{
    "7a44476d.a179c8": {
        "user": "${MQTT_USER}",
        "password": "${MQTT_PWD}"
    },
    "850bd469.ceb218": {}
}
```

You can see the environment variables, which will be substituted at runtime for the values contained in the environment variables.

4.4.5 Step 5. Testing the environment variable substitution

If you were running locally you would need to set the environment variables before Node-RED is started, so they are available when Node-RED loads and runs the flow. However, as we are running from a container we need to provide the environment variables as the container is started.

As there are quite a few environment variables that need to be set, so we will switch to reading the environment variables from a file.

1. In a command or terminal window, ensure the current working directory is your Node-RED project directory (`cd [user home directory]/NRdata/projects/Node-RED-Docker`)
2. Create a new file called env.list in your project directory
3. Add the following to the env.list file:

```
NODE_RED_ENABLE_PROJECTS=true
MQTT_CLIENT_ID=nodeRED
MQTT_HOST=mqttBroker
MQTT_PORT=8883
MQTT_USER=mosquitto
MQTT_PWD=passw0rd
MQTT_CA_CERT=/mosquitto/certs/mqtt_ca.crt
WWW=123
```

4. Restart Node-RED:
    - `docker stop mynodered`
    - `docker rm mynodered`
    - Run the appropriate command for your operating system (replacing *YOUR-USERNAME* with your own username):
    - **Windows**:

      `docker run -itd -p 1880:1880 --env-file env.list -v c:\Users\YOUR-USERNAME\NRdata:/data -v c:\<full path to where moreNodeREDWorkshop repo cloned>\moreNodeRedWorkshop\en\part5\broker:/mosquitto --network NRbridge --name mynodered nodered/node-red` * **Mac OS**:

      `docker run -itd -p 1880:1880 --env-file env.list -v /Users/YOUR-USERNAME/NRdata:/data -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mynodered nodered/node-red` * **Linux**:

      `docker run -itd -p 1880:1880 --env-file env.list -v /home/YOUR-USERNAME/NRdata:/data -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mynodered nodered/node-red`

    e.g. on mac, logged in as user brian, the command might look like:

    `docker run -itd -p 1880:1880 --env-file env.list -v /Users/brian/NRdata:/data -v /Users/brian/moreNodeRedWorkshop/en/part5/broker:/mosquitto --network NRbridge --name mynodered nodered/node-red`

4.4.6 Step 6. Updating Docker container in dockerhub

To create a containerised version of the latest version of the Node-RED application you need to rebuild and push the updated Node-RED application.

1. Go to the GitHub integration tab in the Node-RED editor and commit and push the latest version of your flow
2. Ensuring you are in the project directory in a command or terminal window, build the container with the command (replace YOUR-DOCKER-USERNAME with your docker username):

   ```
   docker buildx build --platform linux/amd64,linux/arm64,linux/arm/v7 -t YOUR-DOCKER-USERNAME/node-red-docker-sample --push .
   ```

3. Stop the existing Node-RED docker instance and remove it. You cannot have 2 instances of a Docker container with the same name, so if the previous instance of the dockerNR container still exists, then you need to remove it before you can deploy a new instance:
   - `docker stop mynodered`
   - `docker rm mynodered`
   - `docker rm dockerNR`
4. Pull the container image from dockerhub to ensure you have the latest version locally (replace YOUR-DOCKER-USERNAME with your docker username):

   ```
   docker pull YOUR-DOCKER-USERNAME/node-red-docker-sample:latest
   ```

5. Run the new container. For docker you can use the **--env-file** option to pass in environment variables using the env.list file created for the previous step. This time we will use a fully qualified path to the file, so you don't need to be in the directory containing the env.list file. Start your containerised Node-RED application. You also need the container on the NRbridge network bridge to be able to access the MQTT broker container (replace the path to the moreNodeRedWorkshop directory and YOUR-DOCKER-USERNAME with your docker username):
   - **Windows**:

     ```
     docker run -dit --env-file c:\Users\YOUR-USERNAME\NRdata\projects\Node-RED-Docker\env.list -v c:\<full path to where moreNodeREDWorkshop repo cloned>\moreNodeRedWorkshop\en\part5\broker:/mosquitto -p 1880:1880 --network NRbridge --name dockerNR YOUR-DOCKER-USERNAME/node-red-docker-sample:latest
     ```

   - **Mac OS**:

     ```
     docker run -dit --env-file /Users/YOUR-USERNAME/NRdata/projects/Node-RED-Docker/env.list -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto -p 1880:1880 --network NRbridge --name dockerNR YOUR-DOCKER-USERNAME/node-red-docker-sample:latest
     ```

   - **Linux**:

     ```
     docker run -dit --env-file /home/YOUR-USERNAME/NRdata/projects/Node-RED-Docker/env.list -v <full path to where moreNodeREDWorkshop repo cloned>/moreNodeRedWorkshop/en/part5/broker:/mosquitto -p 1880:1880 --network NRbridge --name dockerNR YOUR-DOCKER-USERNAME/node-red-docker-sample:latest
     ```

   Notice:
   - you need to provide the values for content in brackets in the above command : **< >**
   - all the environment variables are set with the **--env-file** option and the env.list file
   - the directory containing the certificates is mapped to a local directory **/mosquitto** within the container using the **-v** option. The **MQTT_CA_CERT** environment variable references the root certificate authority certificate from within this directory.
   - The containers are able to communicate and find each other as they are using the user-defined bridge **NRbridge** using the **--network** option and the **--name** option to name the container instance and enable the container name resolution. The **MQTT_HOST** environment variable is set to **mqttBroker**, so requires the MQTT container to be named **mqttBroker**.

4.5 Summary

In this tutorial you:

- learned how to access environment variables in Node-RED
- replaced node config setting with environment variables
- set environment variables when creating a Docker container
- how to get docker containers to communicate using a Docker user-defined bridge

In this tutorial we used the Docker command line to run containers, but there are higher level services, which make managing containers and their configuration more robust and scaleable, such as Kubernetes.