

Dear reader,

Thank you for taking a look at my tech writing samples!

This doc contains examples of documentation I wrote for previous jobs (there's nothing proprietary here). These article stubs show my ability to break down problems into readable summaries and to write clear reference material.

Everything here is the result of on-the-job investigations as a DevOps engineer solving build and deployment issues. After I found a solution to a recurring problem or discovered a "gotcha", I would often create internal documentation in my team's Confluence space detailing the issue. These docs served as technical notes for my team and enabled developers to solve build pipeline issues themselves by referencing my writing.

If you like what you read here, I'd love to hear from you about how I can help with any technical writing needs you may have!

With regards,

Matt Douston

## Contents

<b>Bamboo Troubleshooting</b>	<b>2</b>
<b>Fix Unable to Detect Changes in Bamboo</b>	<b>3</b>
<b>Restart a Remote Bamboo Build Agent Safely</b>	<b>4</b>
<b>Unmergeable Conditions in WorkZone</b>	<b>4</b>
<b>Fix NPM Unable to Resolve Dependency Tree</b>	<b>5</b>
<b>Waking Up Salt Minions</b>	<b>5</b>
<b>Sonar Qube Set Up for Angular.JS</b>	<b>6</b>
<b>Set your repository's master branch as the default main branch in Sonar Server</b>	<b>7</b>
<b>Enabling Sonar Qube Code Coverage</b>	<b>7</b>
<b>Sonar Qube Set Up for .NET</b>	<b>8</b>
<b>Octopus Retention Policies</b>	<b>10</b>
<b>Octopus Variables</b>	<b>11</b>
<b>Troubleshooting Octopus Deployments</b>	<b>12</b>

# Bamboo Troubleshooting

## Jobs queue indefinitely

Check the logs on the build server at E:\Atlassian\bamboo-home\logs. If it says the JVM heap space is full, restart the VM.

The jobs that queue are likely the Docker builds dedicated to the linux remote agent. On that box, when you restart the agent services using the service command (service bamboo-agent.service restart), and then tail the logs in /opt/atlassian/bamboo, you'll see the last log line be something about authenticating the agent with the main server. If the JVM is full on the main server, the log will hang there.

## Jobs immediately "gray out" upon source code checkout

This is when the status banner for a Bamboo build is gray, rather than green for success or red for failure. A gray status usually means the build couldn't start.

Ensure the build working directories, temp directories, log files, or anything that the Bamboo system user might need to use is owned by builduser:linuxuers. Looking at the build failure log should point to the exact file. Sometimes these "grayed out" failures won't produce build logs, but the UI in the build summary should have something you can click for a pop up that shows bamboo system errors.

File ownership can change on remote build agents if someone logs in as root and, as the root user (or some other user), manually runs the agent startup scripts. That can change file ownership from the bamboo user to the user that ran the scripts.

## Files cannot be deleted because their names are too long

node\_modules folders of npm builds can fill up with 50k+ items nested in \built\built\built\built\ directories 16 or more layers deep. Sometimes this folder can contain ~500,000 items with file names far over the OS limit.

You can't delete a file whose name is too long, not even with PowerShell. You first have to move a part of the directory closer to the E:\ drive to shorten the file name, then do a mass delete.

The PowerShell below iterates through the build working directories and moves all node\_modules directories to the root of the E drive. From there, you can manually delete them.

```
$counter=1 #Appended to moved dir names to make them all different.

Get-ChildItem E:\Atlassian\bamboo-home\1 -Directory -Recurse | Where-Object { $_.PSIsContainer -eq $true -and $_.Name -match "node_modules" } | ForEach-Object { $counter++; Move-Item $_.FullName E:\node_modules$counter }
```

# Fix Unable to Detect Changes in Bamboo

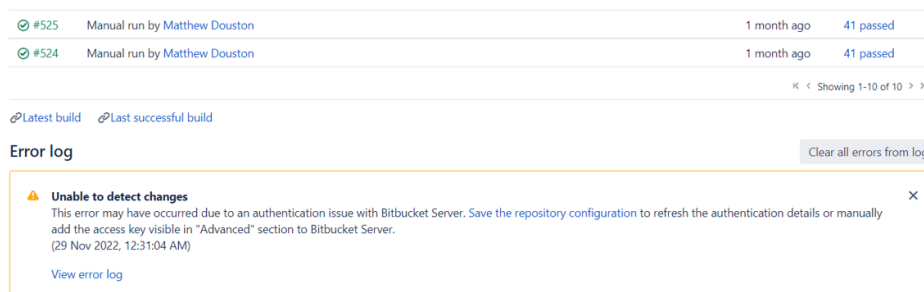
## Quick Instructions:

1. Go to **Plan Configuration** → **Repository Settings**.
2. Select the repository from the list. It may take a moment for the settings to show up.
3. Scroll down and click **Save**.
4. You can now run a build.

## Explanation

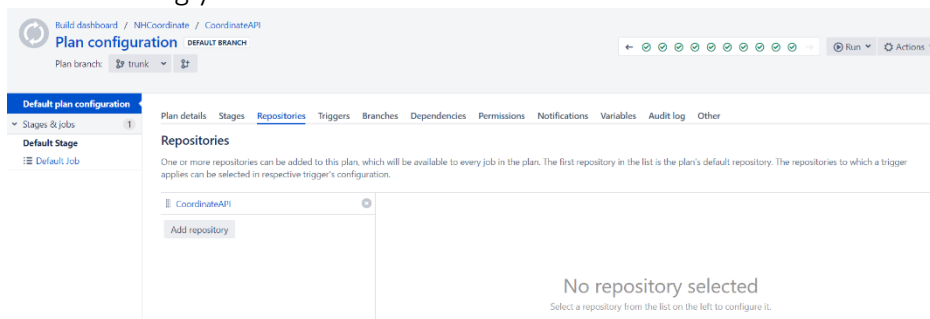
Bamboo plans sometimes need to re-authenticate themselves with Bitbucket. That's why they say *unable to detect changes* at the bottom of the build summary page and why they'll "grey out" if you try to build them.

Bamboo is helpful enough to tell you what to do in this case. A yellow box will show up on the summary page with a *save the repository configuration* link that will bring you to that plan's repository settings.



The screenshot shows a Bamboo build summary page. At the top, there are two build entries: #525 and #524, both manual runs by Matthew Douston, completed 1 month ago with 41 passed tests. Below this is a section for the 'Error log'. A yellow message box is displayed with the title 'Unable to detect changes'. The message states: 'This error may have occurred due to an authentication issue with Bitbucket Server. [Save the repository configuration](#) to refresh the authentication details or manually add the access key visible in "Advanced" section to Bitbucket Server. (29 Nov 2022, 12:31:04 AM)'. There is a 'View error log' link at the bottom of the message box.

That will bring you here:



The screenshot shows the 'Plan configuration' page for a plan named 'CoordinateAPI'. The 'Repositories' tab is selected. The page shows a list of repositories with 'CoordinateAPI' as the default. Below the list, there is a message: 'No repository selected. Select a repository from the list on the left to configure it.' The page also includes a sidebar with 'Default plan configuration' and 'Default Stage' options.

Click on the name of the repo configured with this plan, then scroll to the bottom and click **Save**. It might have a red message box saying it can't read the repo configuration. That's all right. Click **Save** again, wait a moment, and you'll get a green *repository saved successfully* message box. If you get a yellow message box warning about needing to update things manually, that's all right too. You can now run a build.

# Restart a Remote Bamboo Build Agent Safely

1. ssh to the remote bamboo agent.
2. Become the bamboo user. You can become root and switch to the bamboo user, or find the bamboo user's password from our password safe.
3. Run `cd /mnt/data1/bamboo/bamboo-remagent1-home/bin/`
4. Run the bamboo agent startup script: `./bamboo-agent start`

It will take a few minutes for the restarted agent to reappear in the bamboo agent configuration page.

## Why does it have to be done this way?

If the bamboo agent service is NOT set up with `init.d` or `systemctl`, you need to manually run the agent's `bamboo-agent.sh` script to start, stop, or restart the agent process.

When you do that, the file ownership of the directories and files the script touches will become whatever your user was when you ran the script. If you become root and run it, then the ownership of the bamboo agent directories and their content becomes "root:root". Later, when the bamboo user tries to access a build directory, it won't have permission. If a build process fails to authorize a source control checkout, it could be due to this issue.

# Unmergeable Conditions in WorkZone

WorkZone is an Atlassian Bitbucket plugin that enables automatic pull request merges. The logic in the plugin can create unmergeable conditions if not set up carefully. Consider the example below:

```
approvalQuota < 100% & groupQuota['review'] >= 2 & requiredBuildsCount >= 1
```

This says some number of approvals, and at least one successful build, are required to auto-merge a pull request. Note `approvalQuota` is *less than* 100%. This means if everyone in the defined group approves before a build succeeds, the merge condition *cannot* be met.

To fix this, use the *equal to or less than* comparison operator:

```
approvalQuota <= 100% & groupQuota['review'] >= 2 & requiredBuildsCount >= 1
```

Now, if everyone in the "review" group approves before a build completes, the pull request will merge automatically when the build succeeds.

# Fix NPM Unable to Resolve Dependency Tree

## The Problem: npm unable to resolve dependency tree

Bamboo builds are using the global node\_modules folder on the Bamboo server to build a project, not the node\_modules folder created in the build working directory for that project.

### Temporary Fix

Delete the build server's global node\_modules folder:

```
C:\Users\bamboo_user\AppData\Roaming\npm\node_modules
```

Once removed, rerun a build.

### Problem Explanation: Do not use -g with npm commands in build plans or build scripts

The bamboo user was looking for packages in its global node\_modules directory, not in the node\_modules directory created inside a build's working directory. The band aid solution is deleting the global node\_modules folder. However, the thing creating the global folder was the first build step of a certain build plan. It had a step called *install typescript* that ran `npm install -g typescript`. The global flag, -g, was the problem. Removing it ensured this script step no longer created the global node\_modules directory.

# Waking Up Salt Minions

Sometimes Bamboo deployments fail because salt minions don't respond to the salt master.

To fix this:

1. Search for "minion did not return" in the deployment log. You'll see lines like this:

```
build 23-Jul-2018 09:45:18 VM-01:
```

```
build 23-Jul-2018 09:45:18 Minion did not return. [No response]
```

2. Copy the server's name in the line above "minion did not return".
3. SSH into the salt master node.
4. Execute the following for each minion that did not respond during the deployment:

```
sudo salt '<server name>' test.ping
```

You should see the server's name outputted with "true" below it, indicating the minion has responded and is awake. If the minion still does not return a "true" response, rebooting the minion VM often resolves the issue.

5. Restart the failed bamboo deployment. Sometimes waking up some minions will reveal others that need to be woken too. If this happens, repeat these steps until you no longer see "minion did not respond" in the logs.

# Sonar Qube Set Up for Angular.JS

## Required repository additions

1. Add a file called **sonar-project.properties** to your Angular.JS root directory containing these lines:

```
sonar.projectKey=<ProjectName>
sonar.projectName=<ProjectName>
sonar.sources=src
```

2. Add **sonar-scanner** to package.json:

```
"scripts": {
  "sonar": "sonar-scanner"
},
```

3. Add **npm run sonar** to your build script after the npm install:

```
npm run sonar
```

## How do Bamboo builds send scan data to the Sonar Server?

The build server has a sonar config file at C:\sonar\bin\sonar-scanner\conf\sonar-scanner.properties that contains login information for our Sonar Server. The bamboo user uses this file to log into the Sonar server when running `npm run sonar`.

## Required Bamboo plan additions

The sonar.branch.name property must be set in the sonar-project.properties file, but you don't want to manually change it for every branch. To avoid that, set up a Bamboo plan script step to inject the branch name into the sonar properties file. This script step can go anywhere before the main build script. **It can only be run after a scan of the default has been completed**, otherwise branch analysis won't be set up in the sonar project.

### Power Shell script step

**Step name:** *Inject git branch name into sonar properties file.*

**Interpreter (dropdown):** Windows Powershell

**Script type:** Inline. Paste in the below code.

```
$branchName = "${bamboo.planRepository.branchName}"

$rootAngularDir = "${bamboo.build.working.directory}\<your_project>"

# `r`n is the powershell way of adding a new line to a non .txt file:
$sonarProperty = "`r`nsonar.branch.name=" + $branchName

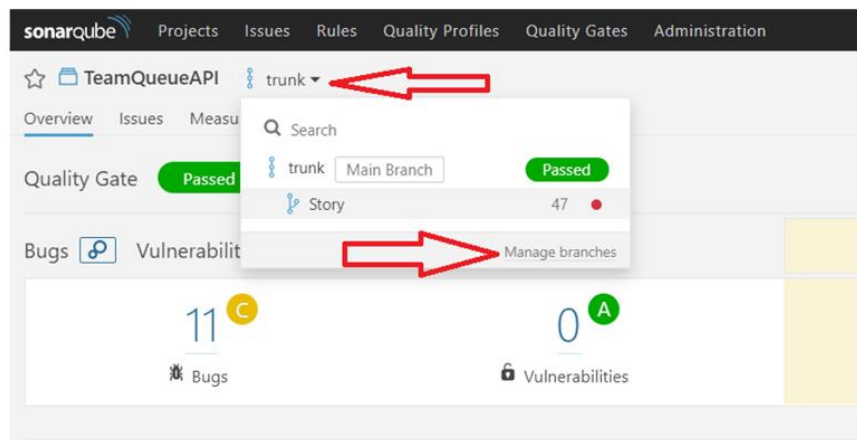
Add-Content $rootAngularDir\sonar-project.properties $sonarProperty
```

## Set your repository's master branch as the default main branch in Sonar Server

By default, the Sonar Server project will set *master* as the default branch. This Sonar Server project is its own entity, distinct from the git repo.

When you later scan a bamboo build that has *trunk* as its default branch, the Sonar Server project will show *trunk* as a branch. It will look like it has two branches named *trunk*.

To fix this, click the branch dropdown at the top of the project page and click **Manage Branches**.



From there, click the gear icon at the right to delete the *trunk* branch. Then rename the Main Branch to "trunk" or "develop" (whichever is the default branch name in your git repo).

Once that's done, the Sonar scan link in the build summary will bring you to *trunk* as the main branch of the sonar project, not to an erroneously named *master* branch.

## **Enabling Sonar Qube Code Coverage**

1) In the build script, add `npm run coverage` to make a "coverage" directory during the build. Make sure this is included before `npm run sonar`!

This requires a line in the package.json file's "scripts" section:

```
"coverage": "ng test --code-coverage=true --watch=false"
```

The "watch=false" prevents the build from hanging on the chrome browser test.

2) In the sonar.properties file, add the following property:

sonar.typescript.lcov.reportPaths=coverage/[lcov.info](https://sonarcloud.io/documentation/coverage/lcov/) (may be deprecated)

OR

sonar.javascript.lcov.reportPaths=coverage/[lcov.info](https://sonarcloud.io/documentation/coverage/lcov/) (if you get warning messages about the above being deprecated)

# Sonar Qube Set Up for .NET

## Sonar for MSBuild

Add the Sonar Scanner for MSBuild- Begin Analysis and Sonar Scanner for MSBuild - End Analysis steps before and after the build step that runs BambooBuild.ps1.

Final tasks Are always executed even if a previous task fails	
Source Code Checkout	✕
Checkout Default Repository	
Script	✕
Create DatePart and Time Part Variables	
Inject Bamboo variables	✕
Import Date Part And Time Part Variables	
Sonar Scanner for MSBuild - Begin Analysis	✕
Sonar Scanner for MSBuild - Begin Analysis	
Script	✕
Run BambooBuild.ps1	
Sonar Scanner for MSBuild - End Analysis	✕
Sonar Scanner for MSBuild - End Analysis	

Add task

Set up the Begin Analysis section:

- Select the MSBuild.SonarQube.Runner.4.10 executable.
- If your .sln file is NOT in the root of your repo, enter the subdirectory path to it in the Working Subdirectory field. Otherwise leave it blank. The scanner has to run in the same directory as the .sln file to work.
- Choose the SonarQube server in the "Choose a SonarQube server for this task" field. The options in that drop down are configured in Bamboo Administration/Add Ons/ Sonar.
- Leave the "Auto Branch" option unchecked for the first scan of your trunk/develop build. This will create a SonarQube project in the sonar server.
- After the initial scan of develop/trunk, check "Auto Branch". This makes it so branch scans show up in a drop down in the project created above.
  - If you don't follow this process, each branch scan will create a separate project in the Sonar server, which will make us hit our 500,000 lines of code limit in our SQ license very quickly.
- Project Key and Project name should be `"${bamboo.planRepository.Name}"`. That's what the SQ project will be named.

Set up the End Analysis section:

- Pick the sonar executable.
- Set working subdirectory if necessary.
- Click "Use one of the configured SonarQube servers from the global administration" and select SonarQube.



Final tasks Are always executed even if a previous task fails

Source Code Checkout

Checkout Default Repository

Script

Create DatePart and Time Part Variables

Inject Bamboo variables

Import Date Part And Time Part Variables

Sonar Scanner for MSBuild - Begin Analysis

Sonar Scanner for MSBuild - Begin Analysis

Script

Run BambooBuild.ps1

Sonar Scanner for MSBuild - End Analysis

Sonar Scanner for MSBuild - End Analysis

Add task

Sonar Scanner for MSBuild - Begin Analysis configuration

Task description

Sonar Scanner for MSBuild - Begin Analysis

☐ Disable this task

MSBuild SonarQube Scanner executable

MSBuild.SonarQube.Runner

Add new executable

The path to the MSBuild SonarQube Scanner executable home.

Environment variables

e.g. SONAR\_SCANNER\_OPTS="-Xmx512m". Separate multiple values by space.

Working subdirectory

nH.UnitedServiceRequest.Service

Specify an alternative subdirectory as a working directory for this task.

☒ Use one of the configured SonarQube servers from the global administration.

Choose a SonarQube server for this task\*

SonarQube

Configure your SonarQube servers in the Bamboo administration section.

☒ Fail build if any errors occur during task execution

This will fail the build if errors occur when invoking the Sonar analysis, e.g. when the credentials are invalid or when a wrong command line parameter is used. Ideally, you want to have this enabled.

☐ Fail build when SonarQube reports analysis errors

Fails the build when SonarQube reports any analysis errors (e.g., when a file contains compile errors) even though SonarQubes overall analysis result might be successful.

☐ Fail build when project fails configured quality gates

You can use this setting if you want to break the build if the project fails the configured quality gates (needs SonarQube version >= 5.3).

☒ Auto branch

Auto branch fills either the "sonar.branch.name" or "sonar.branch" parameter automatically with the Bamboo plan branch depending on the setting below. This is necessary when you intend to use Sonar for code reviews (e.g., with Sonar for Bitbucket Server).

☐ Use legacy branching

Instead of using "sonar.branch.name" available in SonarQube >= 6.7 with the branch plug-in, use "sonar.branch" which results in separate SonarQube projects for each branch.

Project key

\$(bamboo.planRepository.name)

The project key that is unique for each project. Allowed characters are: letters, numbers, -, \_ . and :, with at least one non-digit. If not entered here, the project key from your build files will be taken (e.g., when using Maven, it is equals to "groupId:artifactId").

Project name

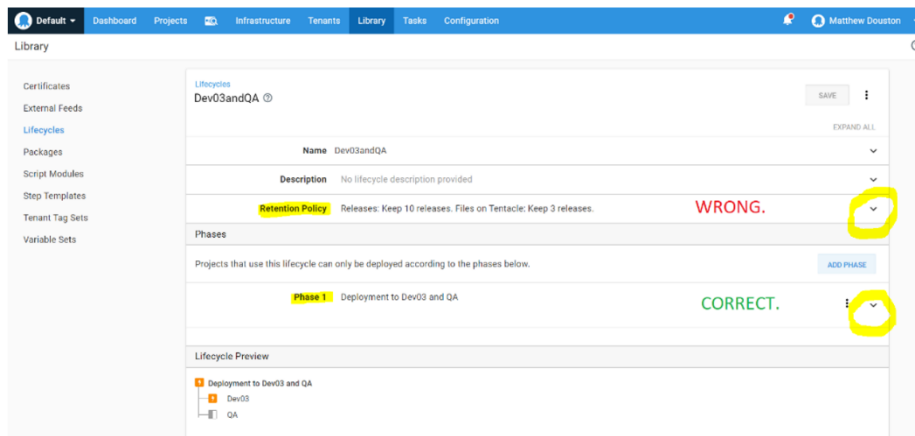
\$(bamboo.planRepository.name)

Name of the project that will be displayed on the web interface. If not entered here, the project name from your build files will be taken (e.g., when using Maven, it is equals to "name").

# Octopus Retention Policies

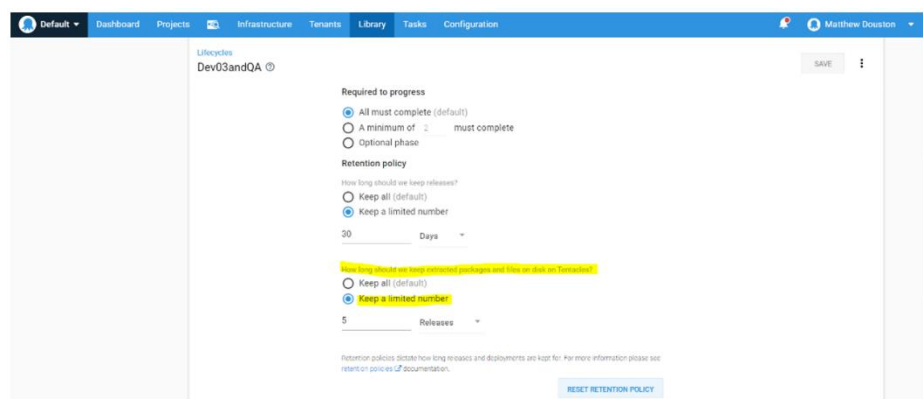
The UI for life cycle configuration is misleading. The retention policies that are applied are NOT the ones configured at the top of the life cycle page.

Look farther down to the **Phase** section and click the expansion arrow. You will see retention policy settings there. Those are the ones that get applied to a machine at the end of a deployment.



The top retention policy is for how many releases to save in the Octopus UI itself. The second one handles how many packages to keep on a deployment target, which is what we want to clear out regularly.

You can freely delete those packages manually on the deployment target machines and not affect anything. Octopus will copy over a package from the Octopus server to the deployment target if it can't find it saved on the target. However, we want to avoid manual package deletions when saved packages fill up hard drive space on a VM. So, we set a low retention policy.



# Octopus Variables

In this Doc:

- Adding a variable to an Octopus deployment
- Adding a variable set to an Octopus deployment
- Creating a variable set

## To add a variable to an Octopus deployment:

1. From the Octopus dashboard, in the “Project name or description” field, search for the name of a deployment project.
2. Click “Variables” in the list on the left.
3. Click “Enter new variable”, give it a value, and scope it to an environment if you need to. If you need to add a new value, click the blue “add another value” below your variable’s info.

**Variable Scoping:** this allows you to have different values for different environments. For example, you can have a different password for Dev, QA, UAT, Staging, and Production environments. To scope a variable value, click “define scope” next to the value and choose an environment. You can ignore the target role and target fields, but you can narrow it down to deployment roles and specific target machines if you really want to.

**Variable Sets:** Groups of related variables. In the Octopus Library tab you’ll find library sets for database passwords, among other things. If you want to use a variable defined in a library set, you must add it to your deployment.

## To add a Variable Set to an Octopus deployment:

1. From the project page, click “Variables” from the list on the left.
2. Click “Library Sets”.
3. Click the blue “Include Library Variable Set” button at the top right. Search for the library set’s name in the pop up and click the checkbox. You can include as many library sets you want this way. Click “save” when you’re done.

## To create a Variable Set:

1. Click the “Library” tab at the top of the Octopus page.
2. Click “Variable Sets” in the list on the left.
3. Click the dark blue “Add New Variable Set” button on the right. Give it a name in the popup.
4. In the page for your new Variable Set, click “Enter New Variable” in the “Name” column, give it a value, and scope it to an environment if you need to. Click “save”. You also have the option of clicking “add to list” and creating another variable. You can create as many variables as you want before clicking “save”.

# Troubleshooting Octopus Deployments

## Deployment stalled due to (error - task not found)

If you see "This script need to wait for task (error - task not found)" in the deployment logs, that deployment will be stuck forever unless you can get that cleared up.

**Solution 1:** restart octopus. Put Octo in maintenance mode, ensure all running tasks finish, then restart the Octopus server via the Octopus manager on the Octopus server.

**Solution 2:** if the issue persists, it's likely the deploy target this is happening on is a linux box. Log onto that box and check the mounted octotentacle home directory's "Work" dir:

```
ls /mnt/auto/home/octotentacle/.octopus/OctopusServer/Work
```

You'll see date stamped directories. Ensuring that Octo is in maintenance mode, delete the most recent directories, at least the one from the current day.

Then disable maintenance mode and retry the deployment.

## Deployment hung, no logs appear on status page

If you don't see logs showing on the deploy status page, it probably means the log file on the Octopus server is ~100mb.

If you wait a few minutes the logs will appear, but log files  $\geq 100$ mb tend to crash the browser after a while.

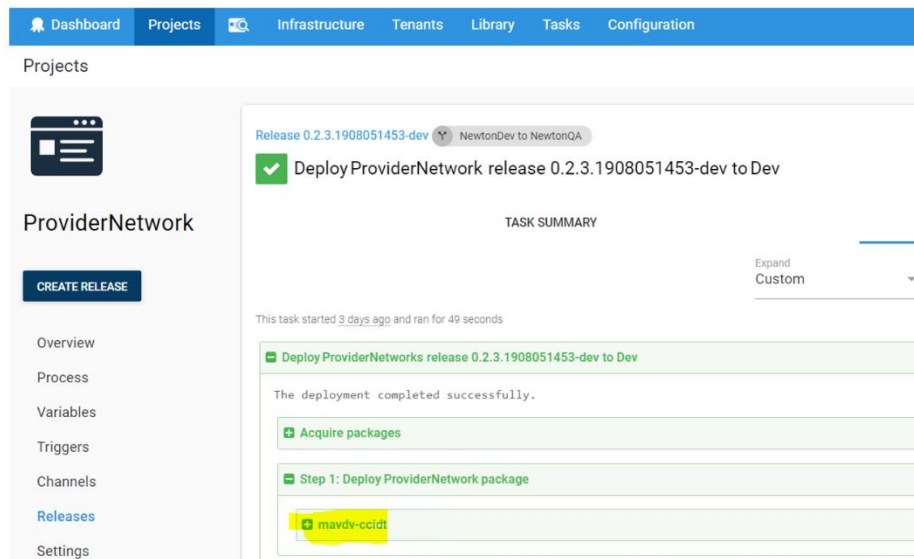
To find the VM the deploy is trying to deploy to, go to the Overview page and click on the deployment beneath the one that's stuck.

The image below highlights the stalled deployment, showing it's in the dev column. We want to click on 0.2.3.1908051453-dev, just below it.

The screenshot shows the Octopus Deploy interface. The top navigation bar includes Dashboard, Projects, Infrastructure, Tenants, Library, Tasks, and Configuration. The left sidebar shows the 'ProviderNetwork' project with a 'CREATE RELEASE' button and a list of links: Overview, Process, Variables, Triggers, and Channels. The main content area is titled 'Overview' and shows a table of deployments. The table has columns for 'Channel', 'Deployment', and 'Status'. The 'Channel' is 'Dev to QA'. The 'Deployment' column shows the deployment name and a status icon. The 'Status' column shows the deployment status and a 'DEPLOY...' button. The deployment '0.2.3.1908071419-dev' is highlighted with a yellow circle, indicating it is the stalled deployment. The deployment '0.2.3.1908051453-dev' is shown below it, indicating the next deployment to click.

Channel	Deployment	Status	DEPLOY...
Dev to QA	0.2.3.1908071511-dev	Aug 7, 2019 3:11 PM	DEPLOY...
Dev to QA	0.2.3.1908071419-dev	Aug 7, 2019 2:19 PM	DEPLOY...
Dev to QA	0.2.3.1908051453-dev	Aug 8, 2019 2:53 PM	DEPLOY...
Dev to QA	0.2.3.1908051453-dev	Aug 8, 2019 3:05 AM	DEPLOY...

Click the line for the deployment step and expand it until you see the name of the VM it deployed to.



ssh into that VM and, if present, delete the `/tmp/Octopus.Calamari.DeploymentJournal.lck` file.

```
[matthew.douston@mavdv-ccidt tmp]$ pwd
/tmp
[matthew.douston@mavdv-ccidt tmp]$ ll
total 32
drwxr-xr-x 2 root      root      4096 Jul  1 10:02 hsperrdata_root
drwxr-xr-x 2 tomcat    tomcatgroup 4096 Jun 28 02:08 hsperrdata_tomcat
drwxr-xr-x 3          1000      4096 Apr 10 2014 IO-Compress-2.063
drwxr-xr-x 3          9307      4096 Apr 10 2014 IO-Compress-Zlib-2.015
drwxr-xr-x 3          500       4096 Apr 10 2014 Net-SFTP-0.10
-rw-r--r-- 1 octotentacle Users    86 Aug  7 13:26 Octopus.Calamari.DeploymentJournal.lck
drwxr-xr-x 3          502       4096 Apr 10 2014 Version-Requirements-0.101022
drwxr-xr-x 4 root      root      4096 Feb 11 2014 vmware-tools-distrib
```

Once that file is deleted the hung deploy should fail, unclogging deployments waiting to deploy to that VM.

That lock file prevents anything on the system from writing to the deployment journal. That file is the octopus tentacle's logs, which the Octopus server tries to write to during a deployment. If the Octopus server cannot write to a tentacle's log file (or if the deploy process the Octopus server is monitoring cannot), the deploy log on the Octopus server will log "thread 1 had lock, but appears to have crashed. Taking lock" over and over, forever, into the task log for the deployment at `C:\Octopus\TaskLogs`. That file is what's populated in the deploy status page in the Octopus web UI.