1. **Bind Mount** can help us to resolve a different kind of problem.
2. **Problem**: Suppose you make a change in your source code; the change will not reflect in the image and container.
   1. You have to stop the currently running container and remove the image, build a new image and then start a new container. Ofs… So lengthy ☹
   2. But of course, **during development**, if you’re using Docker, it would be pretty important to us, that such changes are reflected.

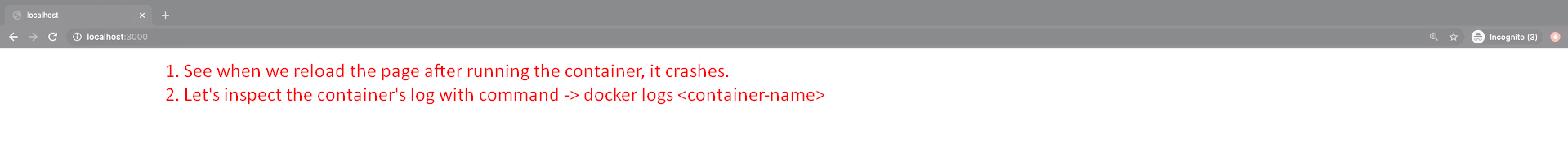
|  |  |
| --- | --- |
| **Volume** | **Bind Mount** |
| Managed by Docker | Managed by Developer. |
| We don’t know the path on host machine for vol. | We Know as we manage. |
| Volume helps us to persist data but no help in editing as we don’t know the path on the host machine. | But with Bind Mount, we can persist data and edit it as we ourselves define the path on the host machine. |
|  | As we know that path, so instead of putting the source code inside an image, we can put the source code inside the **Bind Mount** so that whenever we need to change the source code we can and this way we don’t need to re-create a new image with new changes and start a new container based on this new image. |
| Best for just persistence purpose only. | Best for persistence as well as for editing source code and data. |

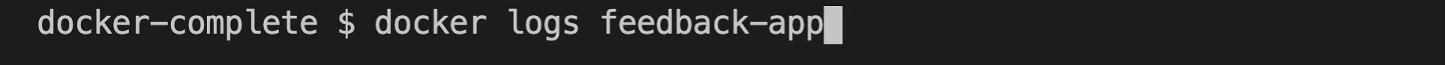
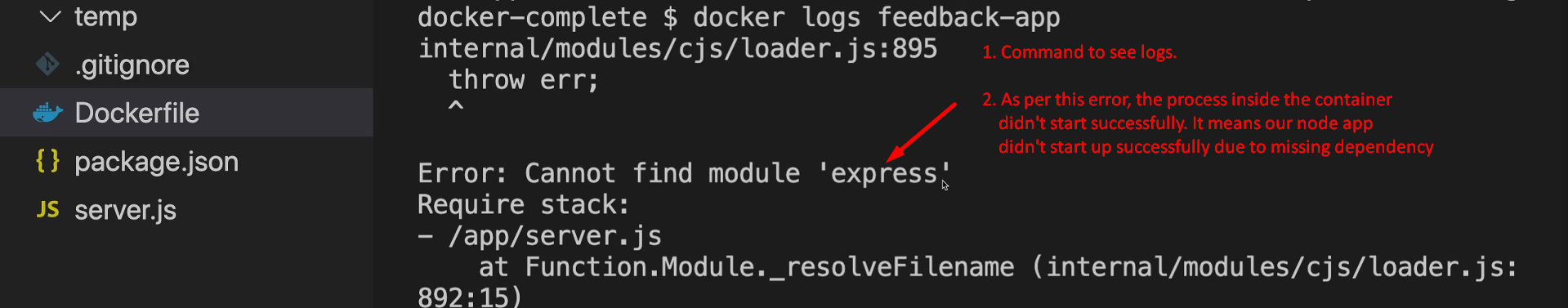
1. How to define a Bind Mount?
   1. As we defined a named volume when creating a container, similarly we create Bind Mount with just a different change.
   2. Let’s define Bind Mount.
   3. Keep in mind that in case of Bind Mount, the path on Host Machine is defined and managed by Developer.
   4. **Syntax**:  
      **PathOnHostMachineWhereYouHaveCodeOrDataFilePathOrFolderPath**:**PathInsideContainerBasicallyOnFileSys**.  
      See, the basic difference b/w the named volume and Bind Mount is in the first part 🡺   
      **PathOnHostMachineWhereYouHaveCodeOrDataFilePathOrFolderPath:** In case of volume, this part is name of the volume ( in case of named volume) and the path is managed by Docker itself.
   5. Now we will map our source code to path inside the container so that if we change anything inside the source code, the reflect is immediately inside the file system inside container.
   6. Text

      Description automatically generated  
      Some correction from the above slide:  
      🡺 The **most path** : The **host path** must be absolute.
2. Okay we have defined the Bind Mount. But make sure that Docker has access to the folder/file on the host machine which you are binding.
   1.  Graphical user interface, text, application, email

      Description automatically generated
   2. If you’re running Docker on window with WSL integration, you can’t see this resource option then no issue as it means no issue with file sharing.
   3. If you’re using Docker Toolbox for Windows then users folder is shared by default and do the following steps to share others.  
      Graphical user interface, text, application

      Description automatically generated
3. When we will run the container, the app inside the container will face an issue.   
   Let’s see and solve it in later lecture.
   1. Text

      Description automatically generated
   2. 
   3. But we ran the above container with –rm so, we can’t see the logs.
   4. Let’s rerun the container without –rm.
   5. Suppose we ran the container without –rm
      1. Text

         Description automatically generated    
            
         Actually, the above error is because our app didn’t start at the first place (The process inside the container didn’t start properly).   
         Definitely this error is caused after we introduced **Bind Mount**.   
         Let’s see in the next lectures how to solve such issues.
4. The following snapshot doesn’t belong to this lecture. In some other lecture, I thought of adding this.  
   