How to build an extension

Running Test Containers

TODO.

We have included .bat files that show the process of building Docker images using our scripts and then running those images as containers. If you having trouble getting the concepts down, try using the Docker Desktop GUI to get a hang of the workflow. After you get what you want running in the GUI, select the options icon next to the container's play button, select 'Copy docker run', and paste that into any text editor to see the command that was used to run the container. This is a helpful way to see some of the MANY docker options in action.

Building Basic Extensions (Docker Compose)

Helpful links:
* https://www.baeldung.com/ops/docker-compose
* https://dev.bostondynamics.com/docs/payload/docker_containers

Extensions are either one image built using their tutorial or a set of docker images. They are composed of (a minimum of 3) essential files.
1. docker-compose.yml
2. manifest.json
3. [extension_name].spx
4. (optional) udev.rules
5. (optional) Additional device files
6. (optional) icon.png

The spot-sdk includes at least one example. See python/examples/spot_detect_and_follow. It includes all the necessary files (and more).

Testing and building are much easier if you have set the BOSDYN_CLIENT_USERNAME and BOSDYN_CLIENT_PASSWORD Windows environment variables to Spot's credentials. This is necessary to auto-login credentials. You can also hardcode them into the Dockerfiles (ENV BOSDYN_CLIENT_USERNAME=[username] / ENV BOSDYN_CLIENT_PASSWORD=[password]), but this won't work when the extension is made and pushed to Spot, only when the container/compose is running locally. We have instead written our BAT files to pass in the GLOBAL environmental variables to set them on the Linux system.

1. Build individual image files for each service you want to run on Spot. See senseable/ping_image/build_docker_ARM64.
   - Make sure global environment variables are set for BOSDYN_CLIENT_USERNAME and BOSDYN_CLIENT_PASSWORD. These are passed into the Linux environment during the container build process.
   - Have to build for ARM64 if on Windows. Docker buildx and docker run lines run something which I believe is a multi-architectural development environment. You then run this and use it to build the ARM64 image.
   - This image is saved as a .tgz file for later packaging together with others. Cab uses senseable/ping_image/run_docker_ARM64 to test this ARM64 image on your Windows PC.
2. Create a new directory for your extension. Copy-paste the image file (.tgz) into it.
3. Add a new blank file named docker-compose.yml to this new directory.
   - This is the core that ties together your Extension. Seems a little leaner than how it's usually used for Docker Compose. The understanding is regular Docker Compose workflows build and cobble the images together all at once, while here it just cobbles together pre-existing images. Ours is lean. Use it to specify where the images are located and other arguments you would normally pass into the command line or Docker GUI when making a container out of an image. Here our image is in the directory already, so we just give it the file name (WITHOUT the .tgz part)
4. Add a new blank file named manifest.json to this directory
   - This JSON file is essentially metadata about your extension. The essential components seem to be description, version number, and images. All are strings, except images which is a list of strings (including file names).
   - You can test this locally on Windows using the 'local_create_and_run_extension.bat' file.
5. Compile this directory into a .spx extension file.
   - This is essentially a .tar with a different file extension name. Pass in the essential files, or just a * at the end to include the entire directory. NOTE: The name of this file will be the name of the extension.
6. Access the CORE IO admin console
   - Select Spot Web Portal > Payloads, then click on the CORE link to edit your CORE IO payload. Then scroll to the bottom and select 'Open CORE I/O Admin Console'. Add your extension there.
7. Run on SPOT and test :)

Writing to Directories Outside of a Container

You'll need to specify the volume of the data directory on Spot. These can also be used to pass data back and forth between containers. In the .yml file, add an additional section for your image that's doing the writing. In the example below, we will have our scripts write to the local directory 'data-directory-in-container' that is inside the container, and these changes will be reflected in the persistent 'data' directory of the CORE IO.

```
volumes:
  - /data:/data-directory-in-container
```

https://docs.docker.com/storage/volumes/
https://stackoverflow.com/questions/66346181/write-files-outside-of-a-docker-container-via-python

https://stackoverflow.com/questions/41648890/why-can-docker-compose-run-create-files-outside-the-container

Diagnosing Errors on Spot

SSH into Spot. Launch your extension via the online portal. Then run 'sudo docker container list' to view current containers. Then run 'sudo docker attach [YOUR_CONTAINER_NAME]' to attach the container terminal to the ssh cmd. This should print Python outputs to the console. TODO mentions the bash.

Misc. Encountered Errors

Attempting to run our network test script resulted in the following error on the container running on Spot during the authentication process:

```
File "/spot_image.py", line 58, in get_image
    bosdyn.client.util.authenticate(robot)
  File "/usr/local/lib/python3.8/site-packages/bosdyn/client/util.py", line 84, in
authenticate
    robot.authenticate(username, password)
  File "/usr/local/lib/python3.8/site-packages/bosdyn/client/robot.py", line 349, in
authenticate
    user_token = auth_client.auth(username, password, self.app_token, timeout=timeout)
  File "/usr/local/lib/python3.8/site-packages/bosdyn/client/auth.py", line 115, in auth
    return self.call(self._stub.GetAuthToken, req, _token_from_response, _error_from_response,
  File "/usr/local/lib/python3.8/site-packages/bosdyn/client/common.py", line 275, in
processor
    return func(self, rpc_method, request, value_from_response=value_from_response,
  File "/usr/local/lib/python3.8/site-packages/bosdyn/client/common.py", line 387, in call
    raise translate_exception(e) from None
bosdyn.client.exceptions.ProxyConnectionError: ProxyConnectionError: The proxy on the robot
could not be reached.
```

It appears this issue is due to SPOT running on external wifi. CORE IO authentication only seems to work when running on Spot's own Wi-Fi network. TODO confirm this and add it to the section above

---

NOTE: If you run into an error related to 'invalid JSON response' on the Extensions Web portal, it likely means you uploaded an extension with an invalid component in the .yml file.
It happened to us when we went from this (function):

```
services:
ping_image:
    image: ping-image-dockerization-arm64
    ports:
      - "21001:21001"
```

To this (an experiment):

```
services:
  ping_image:
    image: ping-image-dockerization-arm64
    network_mode: network_mode: host
    ports:
      - "21001:21001"
```

After rebuilding and re-uploading with the original .yml, everything on the web portal went back to normal.

Revision #5
Created 17 October 2023 16:00:19 by Dwight Howard, II
Updated 17 October 2023 20:11:22 by Dwight Howard, II