Broader support for non-Python apps

Technical Conversation

Author: *Ben Cutler*

Team: *Applications*

Preface

- PBM Card Link: https://atlassian.net/browse/SPEC-2
- Product Requirements Document Link: https://docs.google.com/document/d/1FF-WJnQ-bjv7vZVi_LhbC8U9DMza2Nf2CU2eetTyegY/edit?tab=t.0#heading=h.irewn75eusmk
- Product Manager: Colleen Wilhide

Background

We have gotten large amounts of feedback from users and CFDS that there is strong demand for React apps which we do not support. The apps platform does not support other languages (such as Javascript, or go, etc) because those languages install their dependencies differently than a python app. Rather than require customers build an execution environment bespoke to each package (in the case of JS, the environment would have the required node_modules) or upload a built binary of their app (in the case of a go or Java app) it would make sense to provide users with some more functionality for building their app from source code to something they can run.

References

A huge chunk of this work has been done thanks to Illia in this SPEC-1:
https://atlassian.net/browse/SPEC-1

Rationale

*Currently users of the app platform can upload a `requirements.txt` to their custom apps source version, but that's basically a point solution specifically for users who use Python and manage their dependencies via a requirements.txt file. We could build out a solution to also install a package.json file, but then when users want to have their backend be Go we have support installing `go.mod` files, then when users want to build a go app *and* a react app we need another point solution and things just get dicey and don't scale.*

Rather than build more point solutions, we want to build a more all-purpose approach for users to build their own apps.

We also want to remove the limitation on running apps on the `/apps/<id>` path, while we can run Flask apps via gunicorn, it's an awkward limitation that could cause hangups and confusion.

Out of scope

- *Direct Docker upload apps.*

Stakeholders

- Product: Colleen Wilhide
- Custom Models : Bogdan Klichuk
- Apps: Raul Pineda
- Security: Joe D'Agostino Can you help me assign someone to this?

Logical Architecture

The way the custom model dependency parser is set up, we basically call:

```
Unset
   parser = BaseDependencyParserService.get_requirements_parser(language)
```

Which gets a parser on a per-language basis, and we have a set of supported languages:

https://github.com/datarobot/DataRobot/blob/5e0abf3470f113450107378a4477cca78e29cb56/raptor_lake/dependencies/services.py#L66-L95

Then we call a parser method to generate a big command:
https://github.com/datarobot/DataRobot/blob/5e0abf3470f113450107378a4477cca78e29cb56/raptor_lake/dependencies/services.py#L396-L424

In the case of a custom app with a requirements.txt, this string would be a look something like:

```
Unset
[ "RUN pip install foo=1.2.3 bar==3.4.5 jazz=10.2"]
```

And that would be passed into here:

https://github.com/repo/code.py#L205

A high level overview of the change would be:

1. Instead of saying “If we have a requirements.txt do X”, we say:
    a. “If we have (see Architecture Decisions -> Considered options for interface), do Y
    b. else if we have requirements.txt, do X, (Don’t break the existing flow)
    c. Otherwise just use the default env as is

Architecture Decisions

Editor’s Note: Everything below this headline is optional if you are writing CAD (Conceptual Architecture Design)

Considered options for interface

Option A: Use a `build-app.sh` to compliment the `start-app.sh` script

One option for the interface here is to allow users to supply a build-app script where we have. For a go app with a react frontend, we could have the build-app be something like:

```
Unset
go build .
npm install .
```

And then that binary and node_packages are part of the build.

- [Good] Probably easy for customers to write their own build scripts.
- [NOTE] There is an image cache for the dependency file, and we need to figure that out so we don’t use old image.

Option B: Have users supply a dockerfile

One option for the interface here is to allow a user to make their own dockerfiles

```
Unset
RUN go build . -o myapp
RUN npm install
EXPOSE 5321 (Could we make this work??)
ENTRYPOINT myapp
```

And then that binary is part of the build.

- [Good] Could be SUPER robust and awesome.
- [Good] Would make it much easier to to move customers off of the direct docker image upload
- [Good] Lets customers build more complex / multistage apps which could be really useful.
- [Bad] Could introduce a can of security issues? If you do `FROM` and another customers' image ID, what happens? Can there be Kyverno errors if users mess with the users? If you EXPOSE a port, what does that do (Probably nothing) Maybe this is a solved problem already though since the same issue should exist for execution environments. I'm a little worried this sounds like a cool solution when in reality, we don't get any tangible benefits.

Option C: Extend the existing language tooling

Part of the env flow is: If it's a python app (we know it's python because it is tagged as such on the env), we build the requirements.txt

This could be extended to allow execution envs to be multi-language, and we do a build check for each of those languages.

- [Good] If the docker build step does not have access to the context (eg a package.json) this solution should still work .
- [Good] Probably more secure since users can't run arbitrary code in the build step.
- [Bad] for languages like python, we can do `pip install yada=1.0.0 somethingelse=1.2.3, etc`, but there's no guarantee that approach will work for all languages.
- [Bad] Less functionality available for users.

Decisions

We like Option A the most. Using existing tooling (Option C) requires us to do more work to scale the product, and there's no guarantee we'll be able to support every use case. Option B sounds cool, but if users want to do zany stuff they can build zany execution environments. Requiring a custom dockerfile makes it awkward to do a `FROM` and select your base image, and it makes it hard to develop with.

Considered options for Apps on '/'

Currently, the app ingress exposes `/apps/<app-id>` and that means that your app has to accept responses on that url .

This is because when we send a request to an app, we need to reference a single ingress. If all your apps run on the same domain and the same path, there's no way to specify one app that you want to send requests to.

We have `UVICORN_ROOT_PATH` as an env var which plugs into gunicorn and streamlit, so those work out of the box, but not every user knows about this, and it can be a barrier to adding apps. We would likely need to add some kind of checkbox to the apps (under `resources` most likely) so we don't break existing apps.

It is worth noting that on-prem, we don't need to run apps on `/apps/id` because we use the service URL, which is distinct already. These changes are all for envs where we have LRS ingresses.

Option A: nginx path re-writing

This would involve having the nginx ingress do a path re-write from `/apps/id` → `/`. This is something I have tested, all you need to do is add an annotation to the ingress and you're off to the races

- [Good] Really easy to do!
- [Bad] Couples us more tightly with nginx.

Option B: Apps on Domains

Currently, apps run on a URL like

```
Unset
http://<basic-auth-user>@<basic-auth-pw>@eksX.int.foo.bar.datarobot.com:8080/apps/<id>
```

And instead of having the path (/apps/id) be a way to reference a single app, we would use a domain, such as:

```
Unset
http://<basic-auth-user>@<basic-auth-pw>@eksX.appId.int.foo.bar.datarobot.com:8080/
```

That way we still have distinct URLs, but the domain is what is different.

We will need to adjust the fqdn in the code

However, the logic to get a cluster looks pretty rigid. Maybe we can just say `fqdn = app.id + fqdn` and we're good, but probably not.

- [Good] Would also work!
- [Bad] Might not be technically feasible,
- [Bad] Can't really test on localdev or in a regression env. We need to test on stg which has a slow feedback cycle.

- [Bad] Might be difficult to get certs working if we have a whole bunch of domains
- [NOTE] Talk to Erik L about the issue the apps conductor had with domains.

Option C: Apps on Ports

As mentioned in Option B, apps run on a URL like

```
Unset
http://<basic-auth-user>@<basic-auth-pw>@eksX.int.foo.bar.example.com:8080/apps/<id>
```

We *can* just run each app on its own port:

```
Unset
http://<basic-auth-user>@<basic-auth-pw>@eksX.int.foo.bar.example.com:8081/
http://<basic-auth-user>@<basic-auth-pw>@eksX.int.foo.bar.example.com:8082/
<etc>
```

- [Good] Easy!
- [Bad] If each customer has 20 apps, then we can only support ~3,200 customers (or about 63,000 apps since there are reserved ports)
- [Bad] We shouldn't do this.

Decisions

Option C is not gonna happen. I'm worried that option B (apps on Domains) could cause things to break with SSL certs, or the path not being allowed. Option A I have tested and it will work without causing any risks. If there's a major issue we can move to option B in the future.

Open Issues/Risks/Questions

- [Unknown] Is the build-context from the custom apps source version available when we build the app?
- [Risk] Are there security issues about running arbitrary code / customer code in the build worker?

Observability

Measuring Feature Success

We will see:

1. *Customers building more execution environments*

2. *Customers building + using + sharing more apps*
3. *CFDS making more apps with tools other than streamlit for services and templates*
4. *A new app template for something like React JS which we support.*

Monitoring/Alerting

Apps canary job + integration test should be extended to test this functionality

Data Qualities

Database Indexes and Collections

No new changes to mongo

Data Models

N/A

Data Regimes

N/A

Data Replication

N/A

Data Retention/PermaDelete

We will still be storing images in our image store just like normal. We do not have perma delete set up at the moment.

Data Backup/Restore

Things should work with reverting deleted apps if needed.

Effectiveness

Resources and cost

- *The build effort is probably a single PR*
- *The app's path on / effort is up in the air, but likely not difficult.*

Compatibility

Apps on `/` will need to be a toggleable feature so we don't break existing apps which may be setup to run on `/apps/id`

The build process will need to make sure existing apps still build a requirements.txt if that is supplied.

Scalability

N/A

Security Considerations

- *Is running the build script safe in our environment?*
- *Do dockerfiles have access to undesired things in their docker context, such as the DR source code? Can I `COPY <DataRobot source> .` in a dockerfile, or do a `requests.post` all the files in the container to a remote location?*

Legal Considerations

N/A

Intellectual Property (IP)

N/A

Maintainability: Code Ownership

All code added will be owned by the Apps team.

Milestones

Milestone 1:

Apps on `/` . To mark this as done: we need to test + verify that you can run an app on `/` in ST SAAS (We should test all 4 regression environments + staging)

Milestone 2:

More complex environments. We can mark this as done when we can run a react app on `/` on all 4 regression envs + staging.

Possible Future Work

Based on the current product requirements and proposed technical design, describe any future technical changes or enhancements which are reasonable to expect or imagine for the systems involved in this technical design.

Feedback

Advice

This reflects the outputs from following the Conversations Advice Process. It is here that all advice given is recorded. This ought to include the name of the advice giver, and the date the advice was given. This can frequently take the forms of comments, and if these are provided directly by the advice-giver, they should use RFC 2119 key words to state their opinion.

- MUST
  - ...
- SHOULD
  - Fill out the Technical Specification Process feedback form to help the process improve
- MAY
  - ...

Rejected Advice

Throughout the discussion of a design, various ideas will be proposed which are not accepted. Those rejected ideas should be recorded along with the reasoning as to why they were rejected. This both helps record the thought process behind the final version of the design as well as preventing people from bringing up the same rejected idea again in subsequent discussions.

Template version: 2022-08-12