Managing dependencies of Python pipelines

https://s.apache.org/python-dependency-management-beam-summit-2023

Valentyn Tymofieiev
Google

BEAM SUMMIT
NYC 2023

Managing dependencies is ...

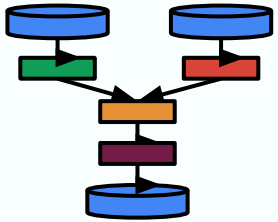
...expressing what pipeline **needs**.

- a Python ML Framework...
- a non-public Python package...
- a third-party Linux software...
- an existing Docker base image for GPU data processing...

... controlling what the pipeline **actually uses**.

- ensuring reproducible, observable, and compatible environments
  - dev environment vs launch environment vs runtime environment

Pipeline launch environment



```
counts = (
    lines
    | 'Split' >> (beam.ParDo(WordExtractingDoFn())
    | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
    | 'GroupAndSum' >> beam.CombinePerKey(sum))
```

```
transforms {
  key: "ref_AppliedPTransform_PairWithOne_9"
  value {
    unique_name: "PairWithOne"
    spec {
      urn: "beam:transform:pardo:v1"
      payload: "\n\305\010\n
beam:dofn:pickled_python_info:v1\032\240\010QlpoOTFBWSZT..."
    }
    inputs {
      …
    }
    outputs {
      value: "ref_PCollection_PCollection_5"
    }
    environment_id: "ref_Environment_default_environment_1"
  }
}
```

python pipeline.py --runner … → pipeline.pb → submit a request to the runner

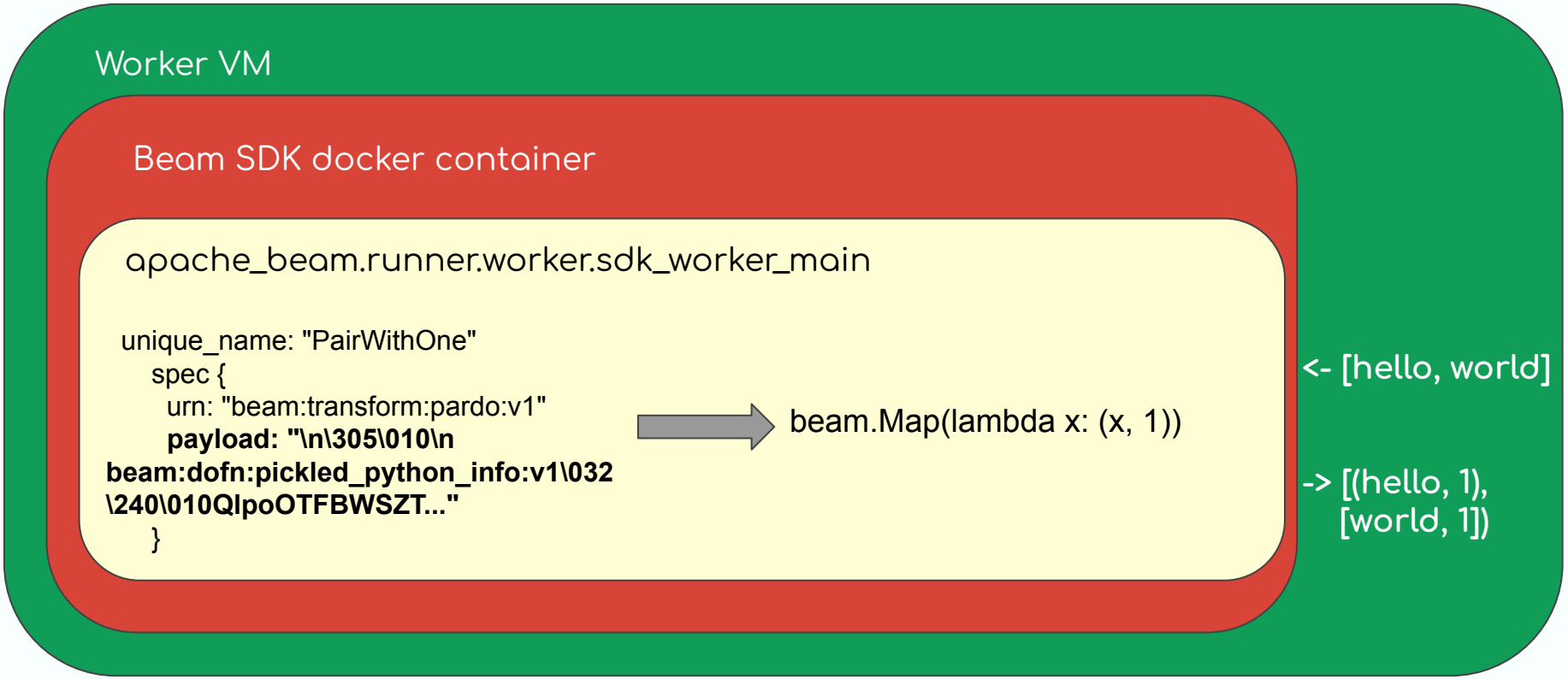
Launch environment

- Used to launch production pipelines
- Keep dependencies to a minimum
  - can even uninstall some Beam dependencies not used on job submission for your pipeline

Dev environment

- Used to iterate on pipeline during development
- Can have additional dev-only deps
  - Jupyter
  - pylint
  - ...

Pipeline runtime environment



https://beam.apache.org/documentation/runtime/environments/

Pipeline runtime environment

- Created by the runner

- Configurable by the user via pipeline options
  - --requirements_file
  - --extra_package
  - --setup_file
  - --sdk_container_image
  - --sdk_location
  - --save_main_session
  - ...

https://beam.apache.org/documentation/sdks/python-pipeline-dependencies/

--requirements_file

- Good for:
  - Supplying a list of dependencies
- Caveats:
  - dependencies are first downloaded locally into a folder under /tmp/<...>/
    path customizable via --requirements_cache="/path/to/cache"
  - entire cache dir is staged to the workers during submission.
    - reduces dependency on Internet/ PyPI on the workers but incurs network cost to stage.
    - --requirements_cache=skip
  - No need to stage what's already in the container image
  - Not recommended for custom containers -- install requirements directly.

--extra_package

- Good for:
  - staging an individual Python package
  - non-public packages

--setup_file

- Good for:
  - Allows submitting a pipeline workflow spanning multiple files
  - Provides a way to install run arbitrary commands on the worker at runtime
    - apt install
  - Removes the need to pass --save_main_session where this is otherwise required
- Caveats:
  - SDK only stages the pipeline package to the runner, but not its dependencies.

--sdk_container_image

- Good for:
  - Complete control over environment
    - Preinstall all pip or apt dependencies
  - Starting side processes
    - See: custom-entrypoint
  - Using custom base image.
    - NVIDIA NGC , Deep Learning Containers, ...
- Caveats:
  - (Dataflow-specific). Large containers:
    - --disk_size_gb=XX
    - --experiments=disable_worker_container_image_prepull

customize all the things!

Using a custom base image

Custom Image + Python + Beam SDK + Beam entrypoint = Custom Beam Container Image.

```
FROM custom_base_image:version
```

```
COPY --from=apache/beam_python3.10_sdk:2.48.0 /opt/apache/beam /opt/apache/beam

ENTRYPOINT=/opt/apache/beam/boot
```

Caveats (for Ubuntu base images):

- Use matching Python version at submission
- apt install python-is-python3
- apt install python3-venv

--sdk_location

- Supply a custom SDK
  - Build your own Python SDK: https://s.apache.org/beam-python-dev-wiki

    ```
    git clone https://github.com/apache/beam.git
    cd beam/sdks/python
    pip install -r build_requirements.txt
    python setup.py sdist
    ```

    ```
    python pipeline.py --sdk_location=./dist/apache-beam-2.48.0.dev0.tar.gz
    ```

  - If using custom SDK builds, you can modify the version.py to 2.48.0+custom
- Disable a self-staging behavior
  - --sdk_location=container

Controlling what pipeline uses

Controlling what pipeline uses

- *I didn't make any changes but my pipeline now fails on startup.*
- *We've upgraded to a new version of Apache Beam but the pipeline started to crash.*
- *I need to recreate a virtual envirionment but when I `pip install apache-beam==<some_old_verison>`, pip takes too long to do dependency resolution*
- The pipeline works well on Direct runner but I am getting a *ModuleNotFound / AttributeError on Dataflow.*

Goals

- Change is good, but make it on your terms.
- Make sure environments are reproducible.
- Have visibility into what has changed.
- Make sure environments are compatible.

Reproducible environments

Change is good, but do it on your terms

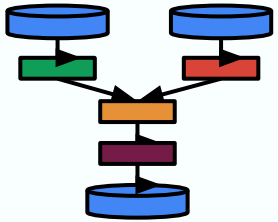
Reproducible environments

Change is good, but do it on your terms

Tools in Python ecosystems for creating reproducible environments

- Requirements files
- Constraint files
- Lock files (Pipenv Poetry, pip-tools)
- Docker container images

Pipeline launch environment

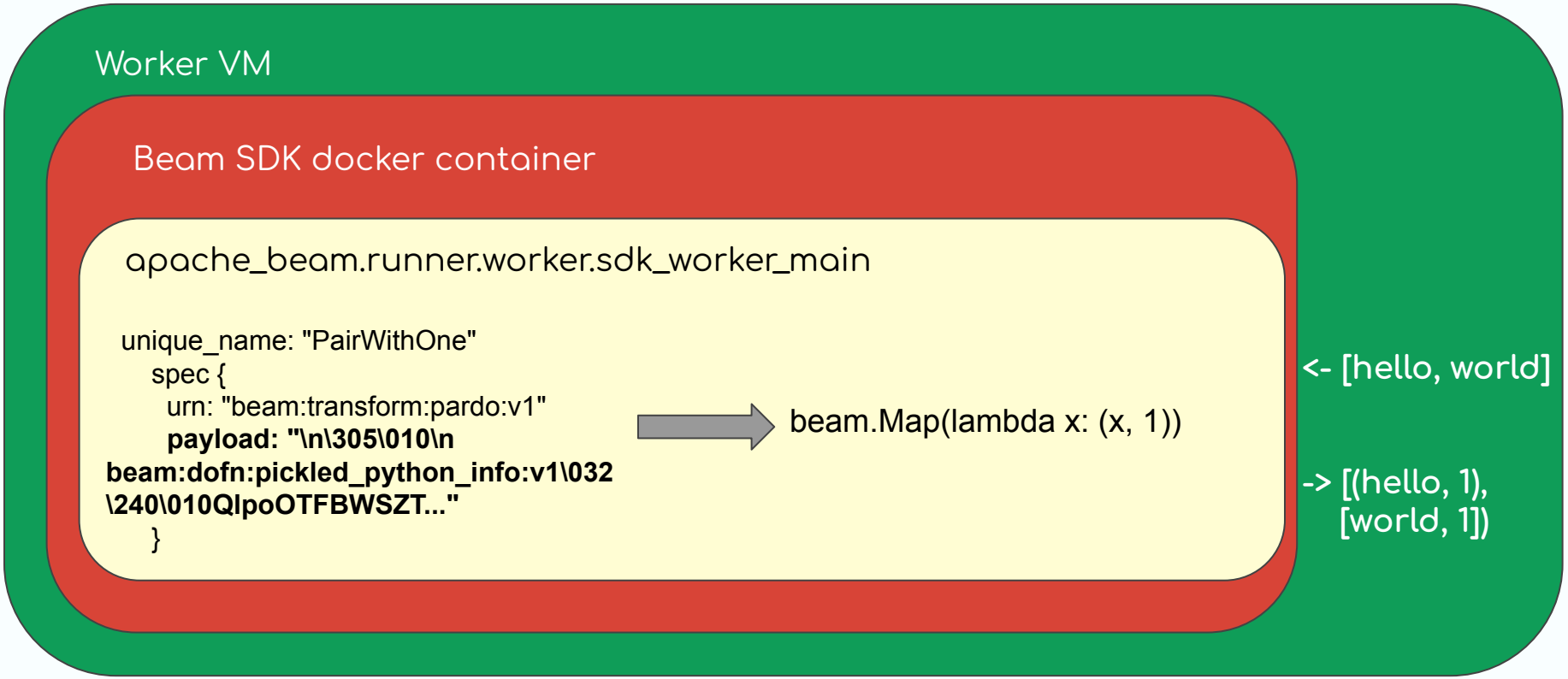


```
counts = (
    lines
    | 'Split' >> (beam.ParDo(WordExtractingDoFn())
    | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
    | 'GroupAndSum' >> beam.CombinePerKey(sum))
```

```
transforms {
  key: "ref_AppliedPTransform_PairWithOne_9"
  value {
    unique_name: "PairWithOne"
    spec {
      urn: "beam:transform:pardo:v1"
      payload: "\n\305\010\n
beam:dofn:pickled_python_info:v1\032\240\010QlpoOTFBWSZT..."
    }
    inputs {
      …
    }
    outputs {
      value: "ref_PCollection_PCollection_5"
    }
    environment_id: "ref_Environment_default_environment_1"
  }
}
```

python pipeline.py --runner … → pipeline.pb → submit a request to the runner

Pipeline runtime environment



https://beam.apache.org/documentation/runtime/environments/

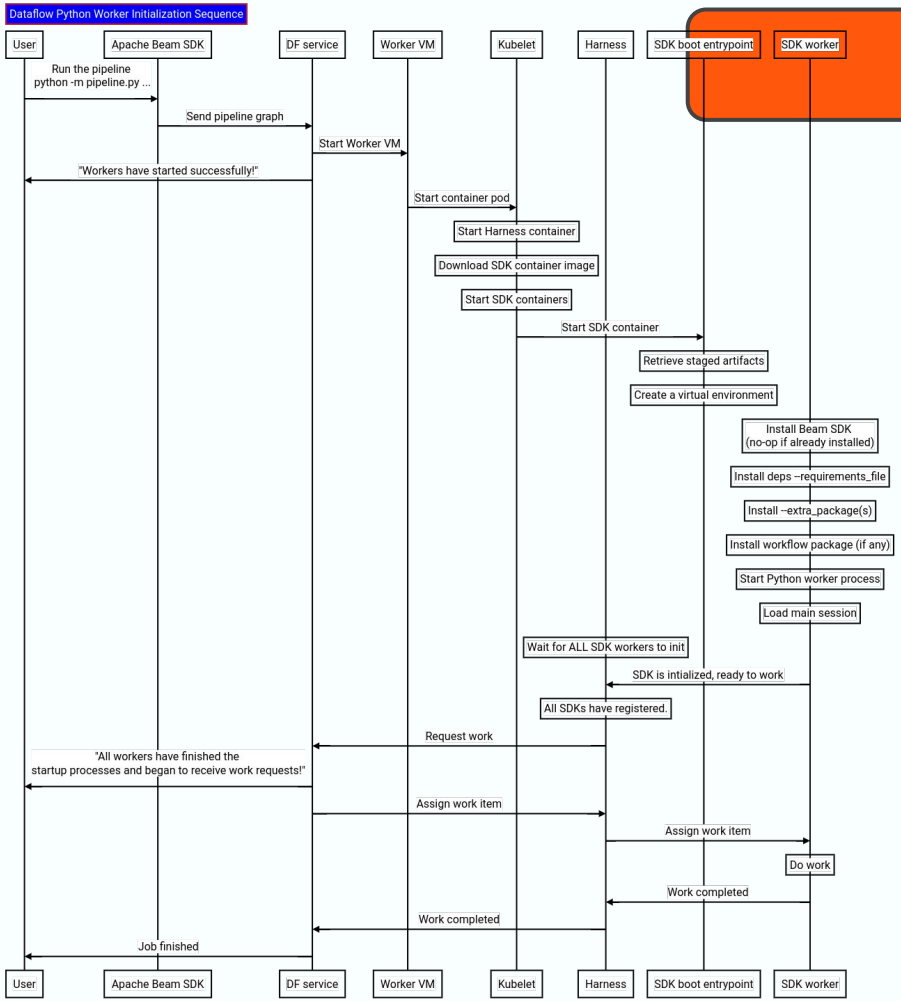
Example: launch environments

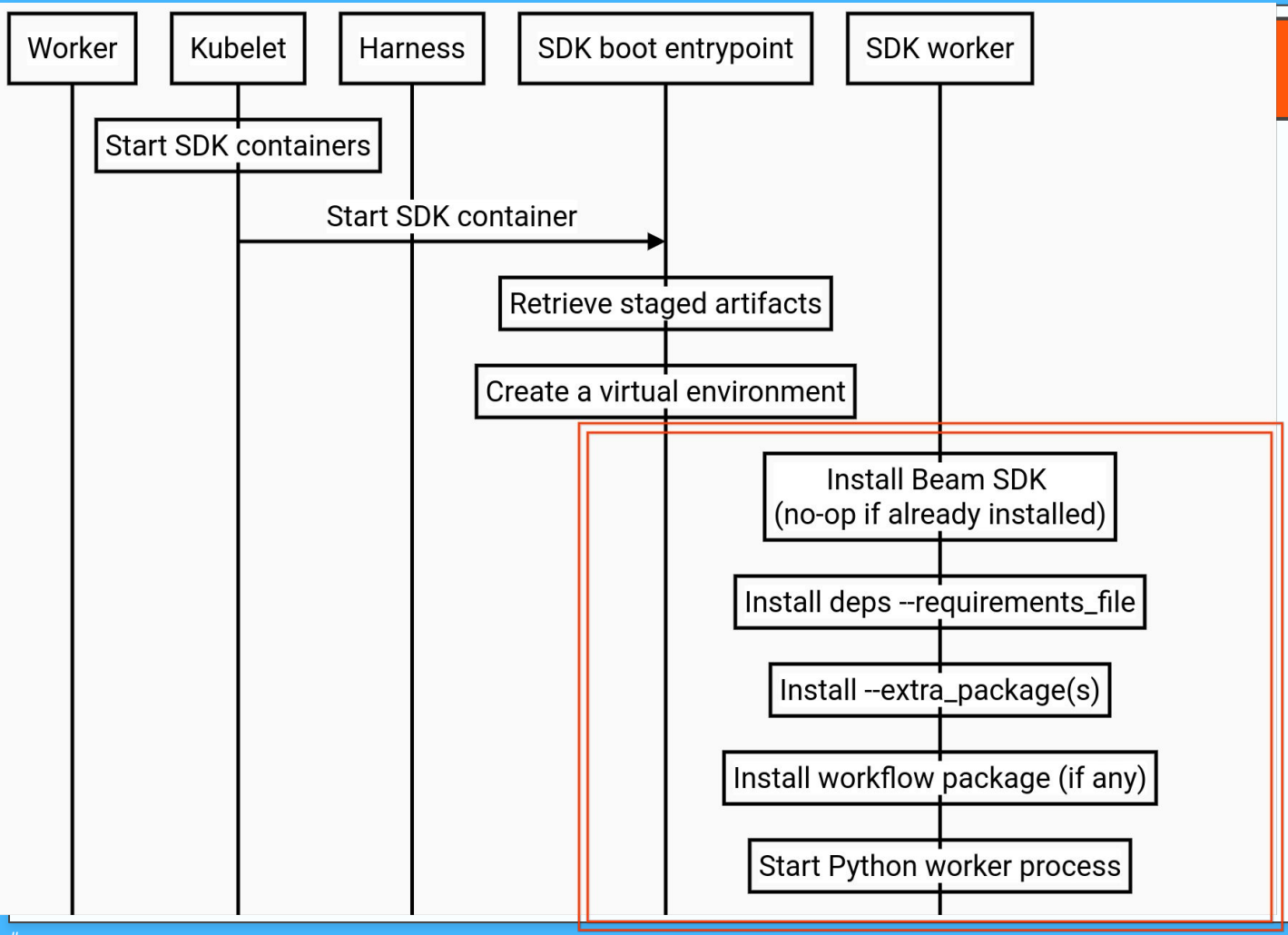
- Install Beam with constraints

```
BEAM_VERSION=2.48.0

PYTHON_VERSION=`python -c "import sys; print(f'{sys.version_info.major}{sys.version_info.minor}')"`

pip install apache-beam==$BEAM_VERSION --constraint
https://raw.githubusercontent.com/apache/beam/release-${BEAM_VERSION}/sdks/python/container/py${PY_VERSION}/base_image_requirements.txt
```

- Flex templates

Runtime environment init




Worker
Kubelet
Harness
SDK boot entrypoint
SDK worker
Start SDK containers
Start SDK container
Retrieve staged artifacts
Create a virtual environment
Install Beam SDK
(no-op if already installed)
Install deps --requirements_file
Install --extra_package(s)
Install workflow package (if any)
Start Python worker process

Reproducible runtime environment

- Is the environment reproducible?
  - Can your pipeline run without access to dependency repos? https://cloud.google.com/dataflow/docs/guides/routes-firewall#turn_off_external_ip_address
  - If you recreate the environment, will it have same deps?
  - If not, will you be able to tell what has changed?

Reproducible runtime environment

- Is the environment reproducible?
  - Can your pipeline run without access to dependency repos? https://cloud.google.com/dataflow/docs/guides/routes-firewall#turn_off_external_ip_address
  - If you recreate the environment, will it have same deps?
  - If not, will you be able to tell what has changed?
- Options
  - use a preconfigured `--sdk_container_image` pipeline option.
  - supply an exhaustive list of pipeline's dependencies in the --requirements_file pipeline option.
    - Additionally, can use --prebuild_sdk_container_engine to perform the runtime environment initialization sequence ahead of the pipeline execution + look up and reuse the prebuilt image via --sdk_container_image option in the follow up if your dependencies don't change.

Are the environments compatible?

beam.Map(lambda x: (x, 1)) -> **payload: "\n\305\010\n beam:dofn:pickled_python_info:v1\032\240\010QlpoOTFBWSZT..."** -> beam.Map(lambda x: (x, 1))

Are the environments compatible?

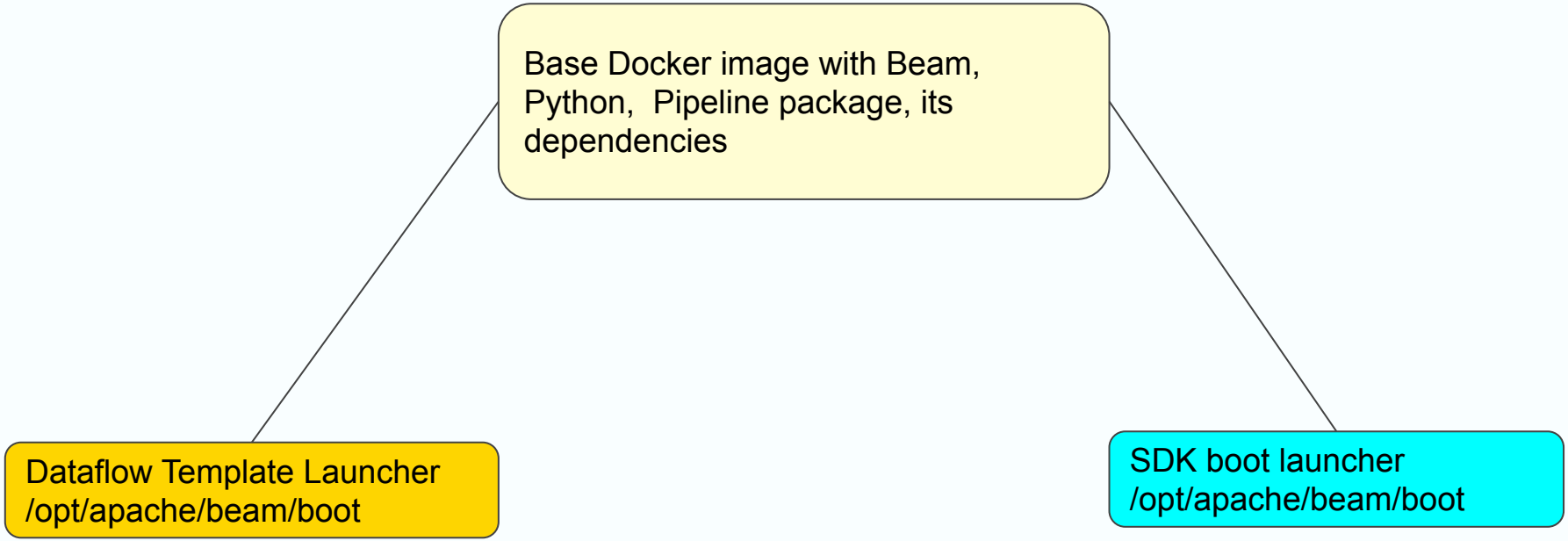
beam.Map(lambda x: (x, 1)) -> **payload: "\n\305\010\n beam:dofn:pickled_python_info:v1\032\240\010QIpoOTFBWSZT..."** -> beam.Map(lambda x: (x, 1))

- pickling library **must** match: dill (or cloudpickle)
  - Compatibility with Beam requirements not as important as matching across envs.
- protobuf must be compatible (better: match).
- Apache Beam version and Python minor version **must** match.
- Libraries used in the pipeline code may need to match (and be available).
  - `from tensorflow.keras import layers`
    - Needs libraries
    - Top level import in a single pipeline file may need [--save_main_session]

So, are my environments compatible?

- Make environments reproducible and observable
  - Compare the diff.
- Better: Eliminate the diff!
  - Install the same requirements
- Better: Use the same environment for submission and runtime:

Same launch + runtime environment



Templated launch environment

Customized runtime environment

See: How to build Flex template from custom custom image

Valentyn Tymofieiev

QUESTIONS?

Email: valentyn@google.com
Github: tvalentyn
Feel free to reach out to share what works, what doesn't.


BEAM
SUMMIT