Pipeline that builds JUST an AMI

https://github.com/mitodl/ol-infrastructure/blob/main/src/concourse/pipelines/infrastructure/docker_baseline/packer_pipeline.py

Probably the biggest thing to remember is that everything is imperative and compounding. That is, things happened in the order they are defined and typically build upon previously defined items.

Line 7: Defines a **resource_type** for 'hashicorp release', because it isn't a built-in with concourse.

Lines 8-27: Defines **resources** for the various items in this pipeline, including hashicorp releases, git_repo releases and a github_release as well. These are all invoking helper functions from lib/resources.py and as such they have different sets of parameters they require depending on what they are for. They are all significantly easier than defining a 'resource' object yourself, though.

Lines 29-34: Making a short list object of **GetSteps** which is made from the *names* of *some* of the resources defined above. This just makes the next step a little easier to read.

GetSteps and PutSteps invoke **Resources directly** and are distinct from **TaskSteps.**

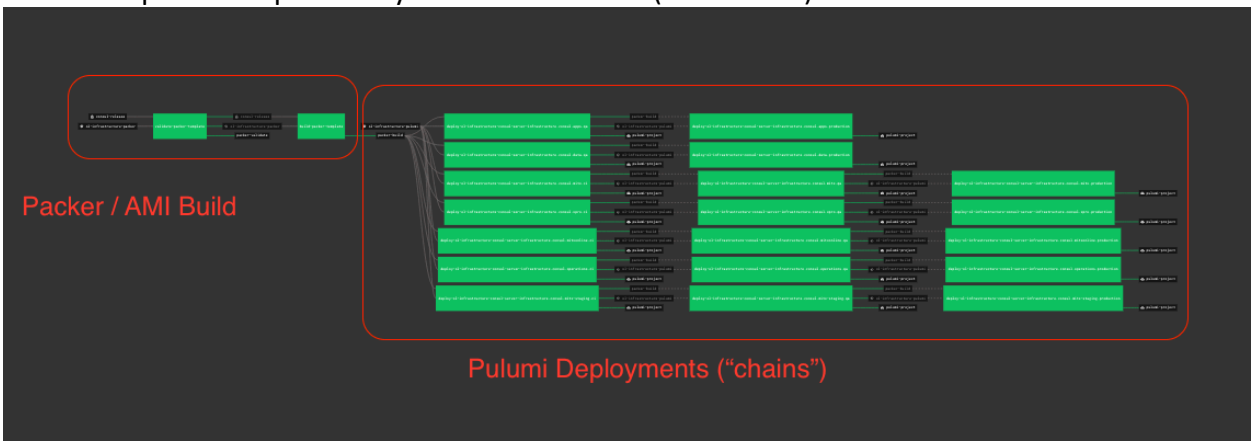
Lines 36-48: Okay now we're going to start doing something. Everything else before was pre-reqs / definitions. This block of code invokes the **packer_jobs()** helper function found here:

https://github.com/mitodl/ol-infrastructure/blob/666ddc4510bfe2d27639f0b3240e9d2fbc95dcd7/src/concourse/lib/jobs/infrastructure.py#L32

This function returns a 'PipelineFragment' object which is exactly what it sounds like, a small part of a pipeline.

This screen shot isn't from this pipeline, but it works for illustrative purposes. The **packer_jobs()** helper function returns the first circled block of pipeline, including the

relationshipts to the previously defined *resources* (lines 29-34).



Packer_jobs() takes very specific arguments relating to this particular activity, validating packer config and then invoking packer to build the AMI, and then returns a pipeline that does that, using the appropriate resources and creating the required **tasks** needed to get that activity done. It is a helper function for this one very specific thing. There are other helper functions for the pulumi steps and other tasks as well.

Lines 50-54: Now we need to make an intermediary pipeline fragment to piece together all of our **resource_types, resources, and jobs** (The three root items in any concourse pipeline definition).

Any pipeline or pipelinefragment has data members representing these three items (lists of these three items, to be specific).

So **combined_fragment** is made up of

1. the **resource_types** from the returned value of **packer_jobs()** joined with a standalone list containing only hashicorp_release_resource (defined on line 7). You’ll note in lines 36-48, hashicorp_release_resource is never passed into the **packer_jobs()**, so if we didn’t include it here in the combined_fragment, we wouldn’t have it in our ultimate pipeline output and it wouldn’t pass the set-pipeline invocation with fly.
2. The **resources** from the returned value of **packer_jobs().** This didn’t need to be joined with anything, because when we invoked **packer_jobs()**, we passed in as parameters all of the resources we are interested in for this pipeline.
3. The **jobs** from the returned value of **packer_jobs().** Again, this doesn’t need to be joined to anything else in this instance because we are only interested in the jobs created by our invocation of **packer_jobs().**

Lines 58-69: Finally, glue it all together as a complete Pipeline object, rather than a potentially incomplete fragment.

This is basically the same procedure as building the pipeline fragment, combining the three core types that make up a pipeline, **resource_types, resources, and jobs.** In this case, we’re pulling

in the resource_types from the combined fragment, because we're sure it includes any resource types returned by **packer_jobs** as well as the one resource_type we defined earlier.

Secondly, we're pulling in the resources from the combined fragment and joining it with a list of ALL of the resource we ourselves defined earlier. This will join together any resources defined by **packer_jobs()** as well with the ones we created.

Finally, we know the only jobs we care about are the ones from the combined fragement, and we know that combined fragment only has jobs that were returned from **packer_jobs()**.

Lines 72-81: These will output the compete pipeline definition to your console as well as to a file in the current working directory **definition.json**. Additionally, it will output a fly command that you can use to apply the pipeline to the production environment + appropriate team.

So something like

```
~/code/ol-infrastructure$  (md/ovs_migration2)| poetry run python
src/concourse/pipelines/infrastructure/docker_baseline/packer_pipeline.py
…
…
    "expose_build_created_by": null,
    "old_name": null,
    "public": null,
    "check_every": "24h",
    "webhook_token": null,
    "icon": "lock-check",
    "type": "hashicorp-release"
   }
 ]
}
fly -t pr-inf sp -p packer-docker-baseline -c definition.json
```