CI/CD Pipelines

JAROD LATTA

SYSTEMS ARCHITECT, VISIMO

Who am I?

Jarod Latta

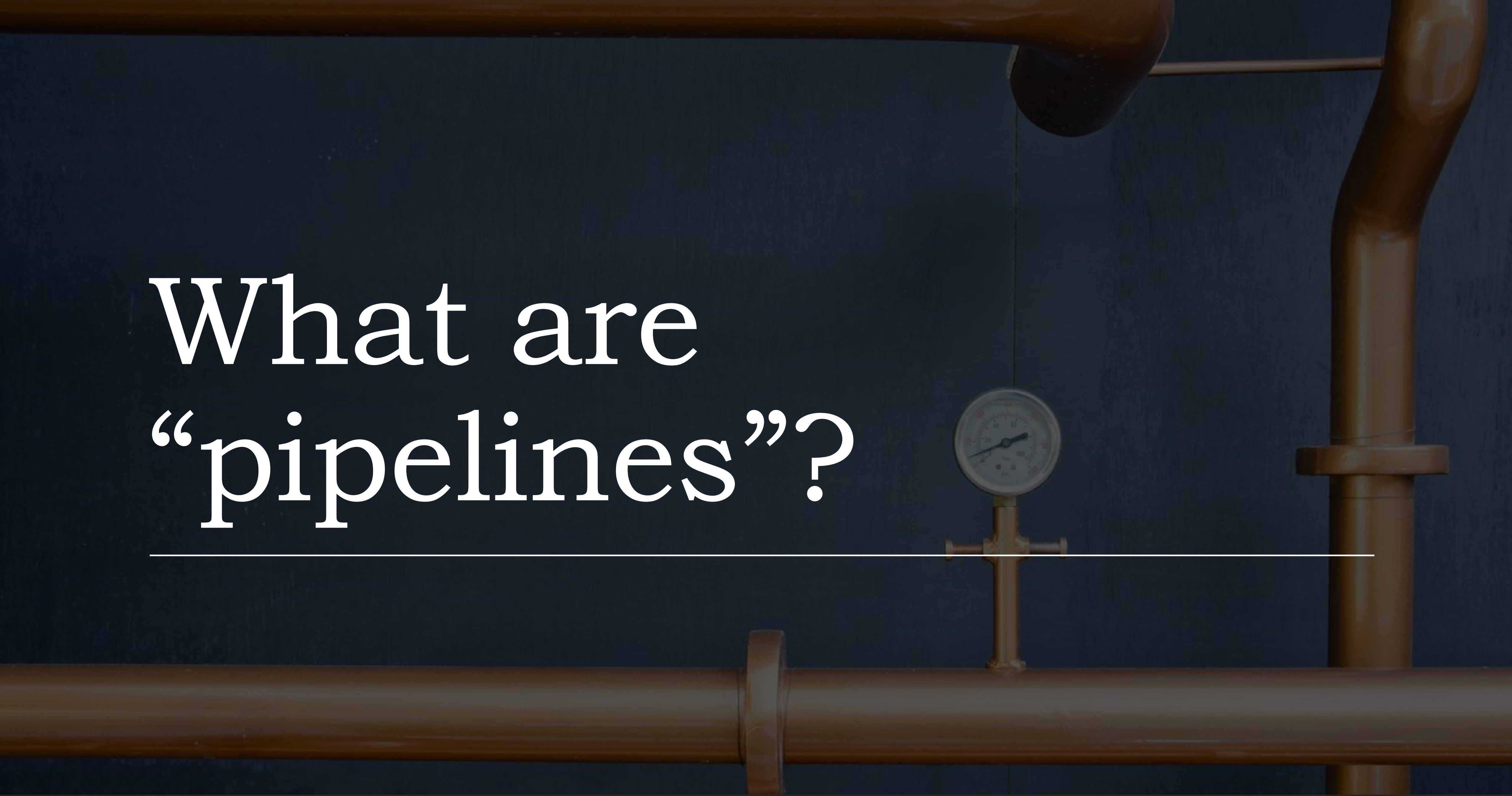
Graduated from Pitt, Fall 2020

BS of Computer Science from SCI

Minor in Linguistics

Certificate for Russian, Eastern European, and Eurasian Studies

jarod@visimo.ai



What are “pipelines”?

Introduction to Pipelines

Formally: “A CI/CD pipeline is a series of steps that must be performed in order to deliver a new version of software.” ~ RedHat.com

Continuous Integration and Continuous Delivery (CI/CD) is a well-defined set of operating principles that enable application code to be delivered more frequently and more reliably.

- The implementation of CI/CD principles is called a **CI/CD pipeline.**
- Guiding idea: determine what must be done, automate it so that you never have to think about it again.

A **step** is a command. Example: `javac *.java`

Software **versions** are the artifact result of a pipeline. They are compiled source code, a built package, a Docker container, etc.

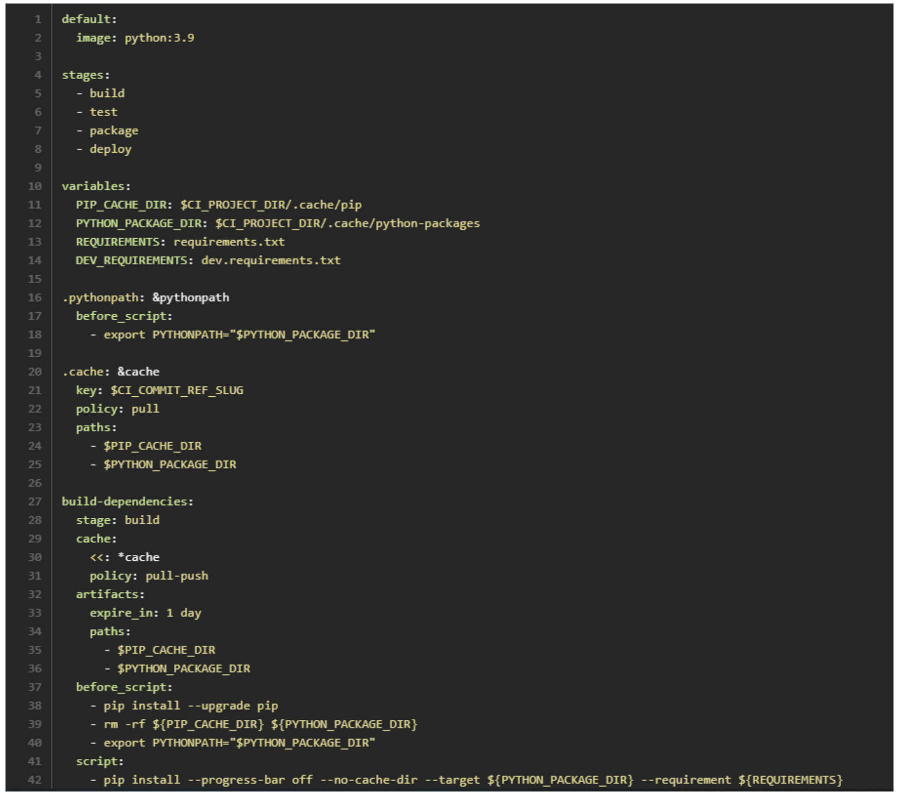
What does a pipeline look like?

In code…

Typically, pipelines are defined in YAML files

YAML is a file format that acts as a *superset* of JSON

- All JSON is valid YAML
- Not all YAML is valid JSON
- Consists of key value pairs
  - Values may be primitives (string, int, etc.) or objects



```yaml
default:
  image: python:3.9

stages:
  - build
  - test
  - package
  - deploy

variables:
  PIP_CACHE_DIR: $CI_PROJECT_DIR/.cache/pip
  PYTHON_PACKAGE_DIR: $CI_PROJECT_DIR/.cache/python-packages
  REQUIREMENTS: requirements.txt
  DEV_REQUIREMENTS: dev.requirements.txt

.pythonpath: &pythonpath
  before_script:
    - export PYTHONPATH="$PYTHON_PACKAGE_DIR"

.cache: &cache
  key: $CI_COMMIT_REF_SLUG
  policy: pull
  paths:
    - $PIP_CACHE_DIR
    - $PYTHON_PACKAGE_DIR

build-dependencies:
  stage: build
  cache:
    <<: *cache
    policy: pull-push
  artifacts:
    expire_in: 1 day
    paths:
      - $PIP_CACHE_DIR
      - $PYTHON_PACKAGE_DIR
  before_script:
    - pip install --upgrade pip
    - rm -rf ${PIP_CACHE_DIR} ${PYTHON_PACKAGE_DIR}
    - export PYTHONPATH="$PYTHON_PACKAGE_DIR"
  script:
    - pip install --progress-bar off --no-cache-dir --target ${PYTHON_PACKAGE_DIR} --requirement ${REQUIREMENTS}
```

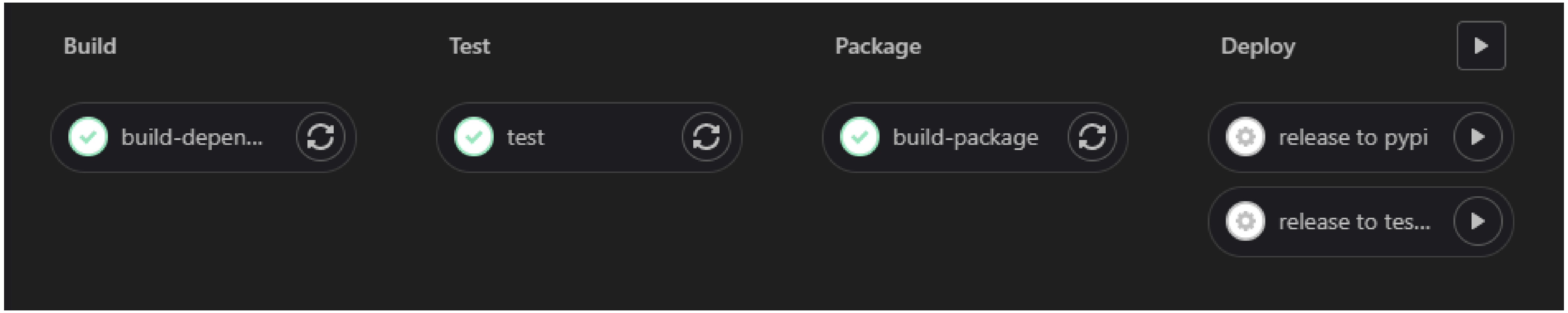
Rendered graphically

Notice we are divided into 4 **stages** and each stage has at least one **job**.

A group of related **steps** (single commands) grouped together make up a **job**.

A group of **stages** make up the **pipeline**.

When two or more jobs share the same stage, they are independent of each other and may run *concurrently.*



How do we choose our steps?

Do it without the pipeline first!

Determine the minimum set of commands to accomplish three tasks:

- **Build** the testing environment from the external dependencies
  - These are Java Packages or Jar files, Python packages, C libraries, etc.
- **Test** the source code in the testing environment
  - In the simplest CI/CD pipelines, this is accomplished with unit tests
  - More complex examples include integration/end-to-end testing
- **Build** the source code into an artifact
  - In this context, an artifact is what the end user should be given access to such as a Jar file or a .class file
- **Publish**/**deploy** the artifact
  - This could be as simple as uploading your source code to GitHub in a .zip file as a “release” or more complicated like uploading to some sort of package registry

So, what do we do with this?