ATLAS Release Tester (ART)

Tülay Çuhadar Dönszelmann, Walter Lampl, Graeme Stewart

CHEP 2019
Adelaide, South Australia
4-8 November 2019

Introduction

- ATLAS Offline Software Release
  - Lives in one large git repository
  - One branch for each release-series (Tier0-production, simulation, development)
  - Code-base is sub-divided into “packages” (feature of our build system)
  - We can defined *Projects* (subset of packages)
    - Example: Event Generation releases use only a subset of packages
- Nightly Builds
  - HEAD of each branch
  - Tagged for each nightly by timestamp
  - Multiple platforms and multiple projects
- Nightly Tests
  - Run for each of the nightly builds
  - Short Tests (executed locally)
  - Long Tests (executed on Grid)

Previous Nightly Testing System

- Run Time Tester (RTT) framework has been used in ATLAS for a long time
  - The system was bound to specific clusters at CERN
  - Depended on AFS
  - Running a single test was not straightforward, because all tests were defined in a single XML file

A new framework for the ATLAS testing system was needed to address these issues

What is ART ?

- ATLAS Release Tester (ART) provides a unified testing system
  - One tool (art command line)
  - One set of tests (for grid or local)
- It allows to submit:
  - Long tests to the GRID
  - Short tests on local machines, to be run in parallel
- It is used by:
  - Automatic Nightly Submission
    - After the nightly release is built
    - Using the gitlab-ci system to manage the submission
  - Users
    - To run jobs locally or on the GRID

Features of ART

- Simple Test Definition
  - Shell or Python tests
  - Adorned with headers to instruct ART
  - Full control by developers
  - Easy to run and reproduce any failure
  - Easy to submit job to GRID
- Predefined set of possible input files (bytestream, simulation, ...)
  - Either on CVMFS or on GRID (rucio)
- Possibility to run post processing
  - Regression tests
  - Histogram comparison
- Automatic download and storage of results

ART Command Line Utilities

- User defines test and adds art-headers in the form of key-value pairs:

```
test_example.sh(.py)
#art-type: grid
#art-input: …
…

<actual test lines go here>
```

# art-type : grid | build	(To run on grid or locally)
# art-include: <String>	(Nightlies the script must run on)
# art-input: <String>	(Name of the dataset to be read in the grid)
# art-nfiles: <Int>	(Number of files to be read from the dataset)

- User run jobs in parallel or submits to GRID using ART

```
art.py run        [options] <script_directory> <sequence_tag> [<test_names>…]
art.py grid       [options] <script_directory> <sequence_tag>
```

waits for grid result to be ready to copy to EOS using ART:

```
art.py copy        <indexed_package>
```

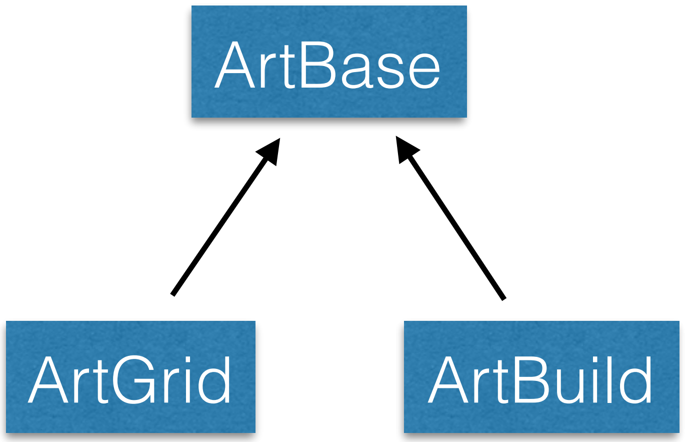
- Some ART Command line utilities (CLU):

```
ART - ATLAS Release Tester.

Usage:
  art.py run              [-v -q --type=<T> --max-jobs=<N> --ci --run-all-tests --timeout=<S> --copy=<dir>]
<script_directory> <sequence_tag> [<test_names>...]
  art.py grid             [-v -q --type=<T> --max-jobs=<N> -n --run-all-tests] <script_directory> <sequence_tag>
  art.py submit           [-v -q --type=<T> --max-jobs=<N> --config=<file> -n --run-all-tests] <sequence_tag>
[<packages>...]
  art.py copy             [-v -q --user=<user> --dst=<dir> --unpack --tmp=<dir> --seq=<N> --keep-tmp] <indexed_package>
  art.py validate         [-v -q] [<script_directory>]
  art.py included         [-v -q --type=<T> --test-type=<TT> --out=<file>] [<script_directory>]
  art.py download         [-v -q --max-refs=<N> --user=<user> --dst=<dir>] <package> <test_name>
  art.py compare grid     [-v -q --max-refs=<N> --user=<user> --entries=<entries> --file=<pattern>... --txt-file=<file>...
--mode=<mode> --diff-pool --diff-root --out=<file> --order-trees] <package> <test_name>
```

ART Implementation

- Simple class hierarchy to handle local and grid jobs
  — Fully written in python
- Helper classes to abstract different functionality for things such as configuration, headers, Rucio:
  — ArtConfiguration, ArtHeader, ArtRucio



- Some scripts to handle different functionality:
  — art.py (main script), art-trigger.py (sending trigger to git-lab-ci), art-share.py (input management)
- ART is on gitlab https://gitlab.cern.ch/art in four projects:

  art-sw: ART software project, Classes and command-line tool

  art-submit: ART grid submission project, receiving the trigger and submitting the jobs

  art-gitlab-ci-runner: Runner images (slc6, cc7, grid and local) for ART

  art-www: ART project web site and asciidoc manual

Automatic Nightly Submission

- Nightly Build triggers the ART gitlab-ci system, which runs through 4 stages:

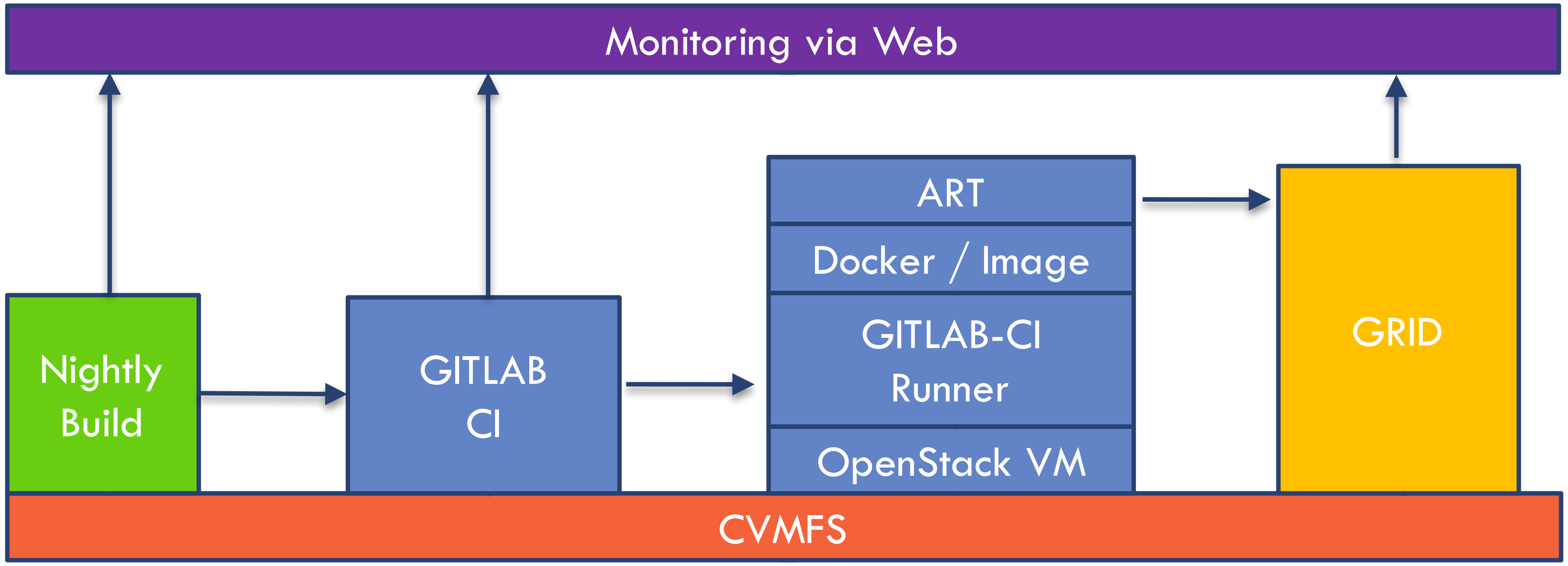
  checkout: Checks out a proper copy of ART

  configure: Verifies if testing is required

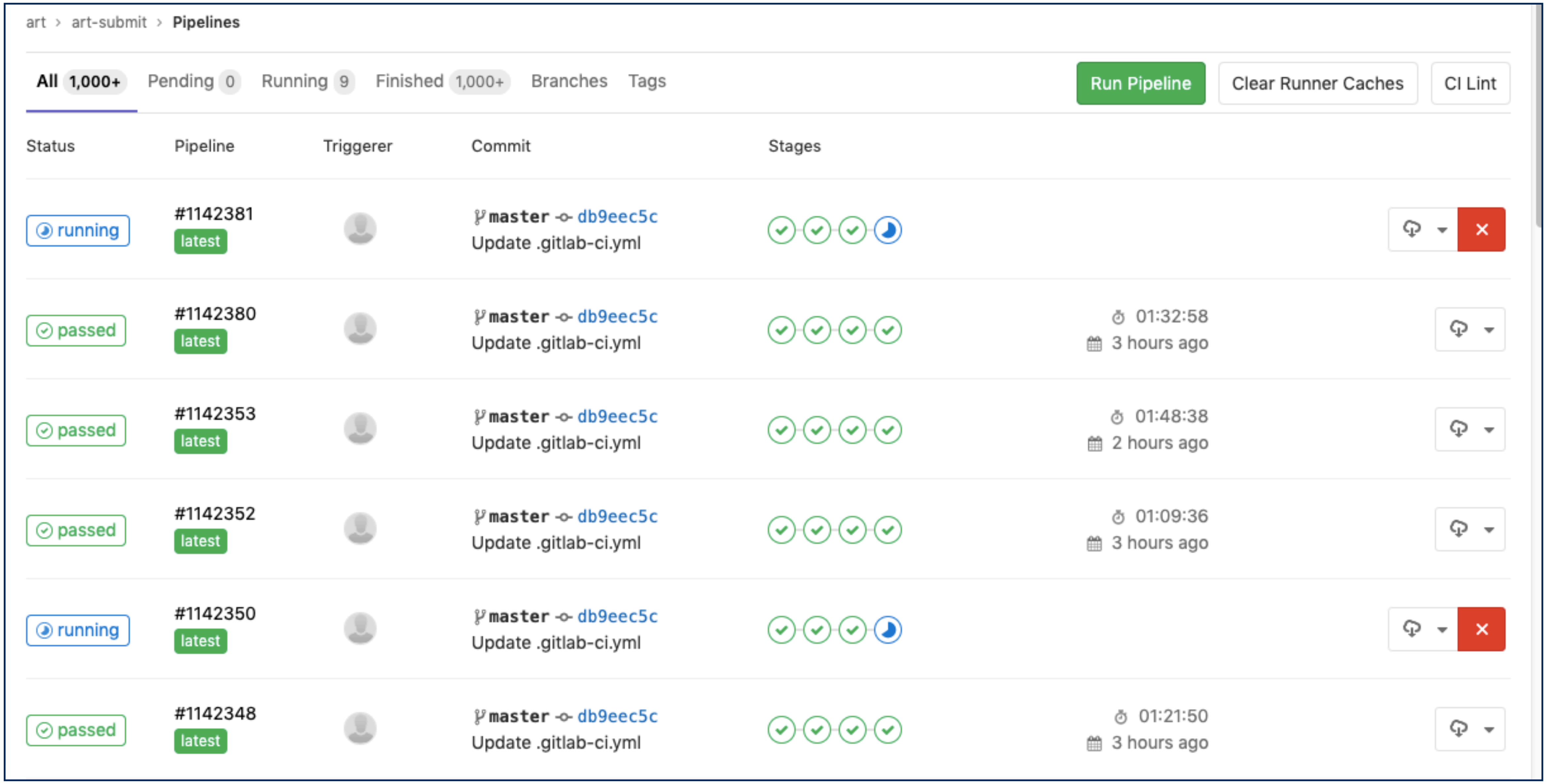
  cvmfs: Verifies the availability of the nightly release on CVMFS (which is distributed to the GRID)

  submit: Submits jobs to grid (ART CLU) and waits for results to be copied

- The 4 stages above run on a set of 5 Virtual Machines for ART, each loaded with docker images to run the ART command line and submit jobs to the grid.
- Jobs can be consulted using a Web Interface looking at either gitlab or GRID output.



art-submit Pipeline



Status	Pipeline	Triggerer	Commit	Stages		
running	#1142381 latest		master db9eec5c Update .gitlab-ci.yml	✓ ✓ ✓ ◔		✕
passed	#1142380 latest		master db9eec5c Update .gitlab-ci.yml	✓ ✓ ✓ ✓	01:32:58 3 hours ago	
passed	#1142353 latest		master db9eec5c Update .gitlab-ci.yml	✓ ✓ ✓ ✓	01:48:38 2 hours ago	
passed	#1142352 latest		master db9eec5c Update .gitlab-ci.yml	✓ ✓ ✓ ✓	01:09:36 3 hours ago	
running	#1142350 latest		master db9eec5c Update .gitlab-ci.yml	✓ ✓ ✓ ◔		✕
passed	#1142348 latest		master db9eec5c Update .gitlab-ci.yml	✓ ✓ ✓ ✓	01:21:50 3 hours ago	

Each job corresponds to submission for a nightly

ART’s own Continuous Integration (CI)

- Unit and Integration Tests for ART try to cover all its code, runs in gitlab-ci in three phases at every commit.

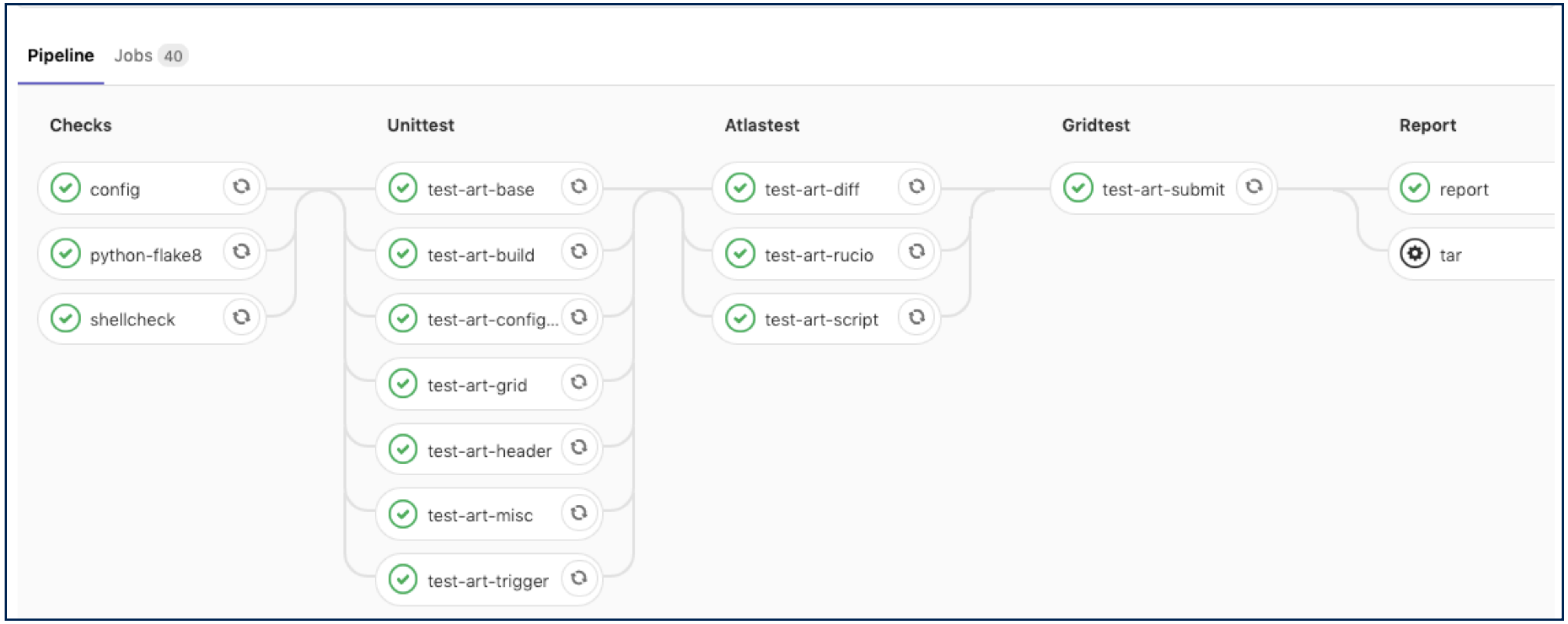
  Checks: ATLAS setup, python-flake8, shell check (30 seconds)

  Unittests: For each of the classes/modules (2 min)

  Atlastests: Local tests to setup and download files (10 min)

  Gridtests: Run when repo is tagged: submit simple job and check results (30 min)

- Coverage: gather all coverage information of unittests and grid-tests and publish
  - ➢ Coverage of the code is around 90%
  - ➢ Test reports per branch available on EOS



Used Technologies

docopt.py: To handle the command-line and its options

yaml and **json**: For configuration and status files

gitlab-ci: To submit nightly tests and wait for their results

open stack Virtual Machines (VM): To run all the gitlab-ci jobs on

docker and docker-images: To have the same environment on all the VMs

BigPANDA: For GRID job submission and monitoring

Rucio: To download results into the VMs

EOS and xrdcp: To copy results back from the VMs into EOS

asciidoc and **asciidocter**:

- To write the ART Manual
- To convert the asciidoc manual to pdf and a website

Summary

- ART is a tool to test the ATLAS offline software
- ART is in production since more than a year now
    - Replacing a system that was bound to legacy infrastructure
- ART continues to evolve depending on the needs of ATLAS and on the evolution of the underlying infrastructure