Pipeline Basics

MDO: HOw To - GitLab - Pipeline Basics

Tags: Party Bus
Categories: Party Bus
14 minute read

Introduction

To create a better understanding of Party Bus' pipeline flow, we have put together the following information for Application Teams and Platform One Teams alike.

Software Versioning

Party Bus uses the most recent version of software whenever available. However, this may impact how individual pipelines work, requiring product teams to make adjustments. Every effort will be made to give product teams advance notice of changes such as these - however, this is a dynamic environment. Product teams should make an effort to run as much of the pipeline locally before any code is committed.

Deployment Branches:
Please note that we did upgrade to the latest version of Gitlab, which requires the use of main, but are still using master for now.

Master vs. Main Branch:
Please note that we did upgrade to the latest version of Gitlab, which requires the use of main, but are still using master for now.

Modifications

Any modification of jobs and stages in the c-ATO pipeline must be approved by MDO.

Jobs and stages of the pipeline will be updated or added when necessary to improve the overall security and quality of images being produced.

Checks - Best Practices

Whenever possible, product teams should test changes locally before making commits. This is helpful to catch obvious errors, minimize excessive commits, reduce usage of shared resources, and test changes more quickly. This is particularly key with the build, lint, and unit test jobs; but can also be done with the dependency check, SonarQube, TruffleHog, build image, and e2e test jobs (the Fortify, TruffleHog, and Anchore jobs cannot be replicated locally).

This guide goes over how to recreate the Dependency Check job locally, but it can be applied to the other jobs, as well.

NOTE:
Pipeline capabilities for checks should be considered a **safety** check, **not** a first check.

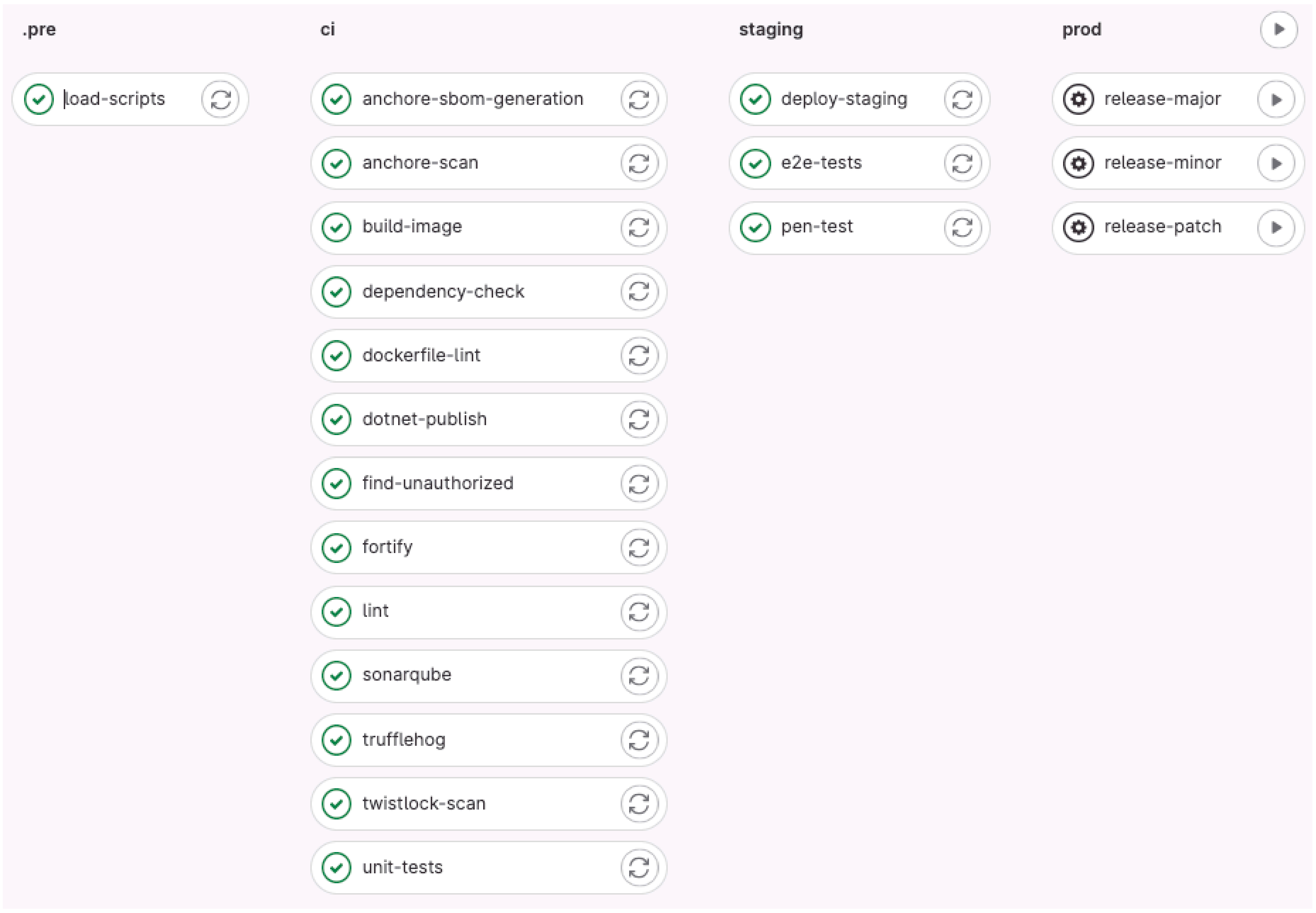
NOTE:
CI/CD pipelines are a tool to automate the software delivery process. Developers are expected to do their due diligence to verify results.

C-ATO Pipelines

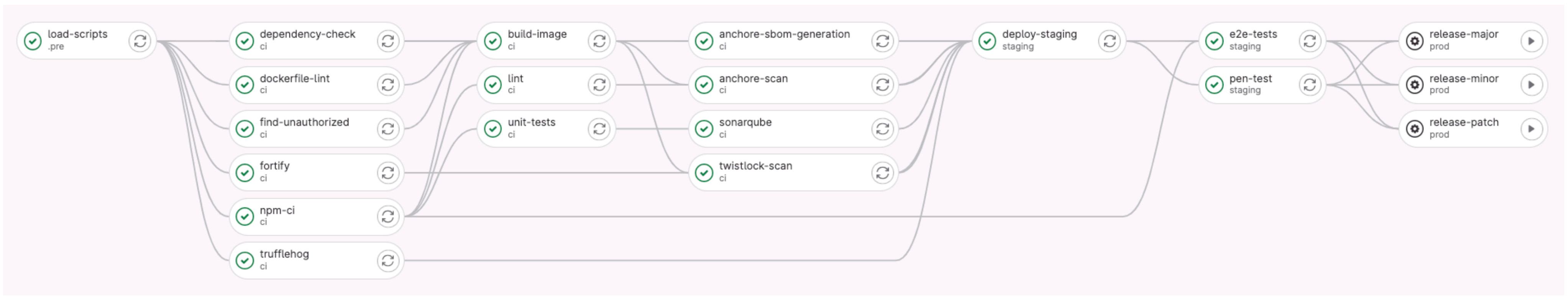
Deployment Pipelines

Deployment pipelines generate a single Docker image to be deployed as part of an application.

The pipeline stages and jobs:



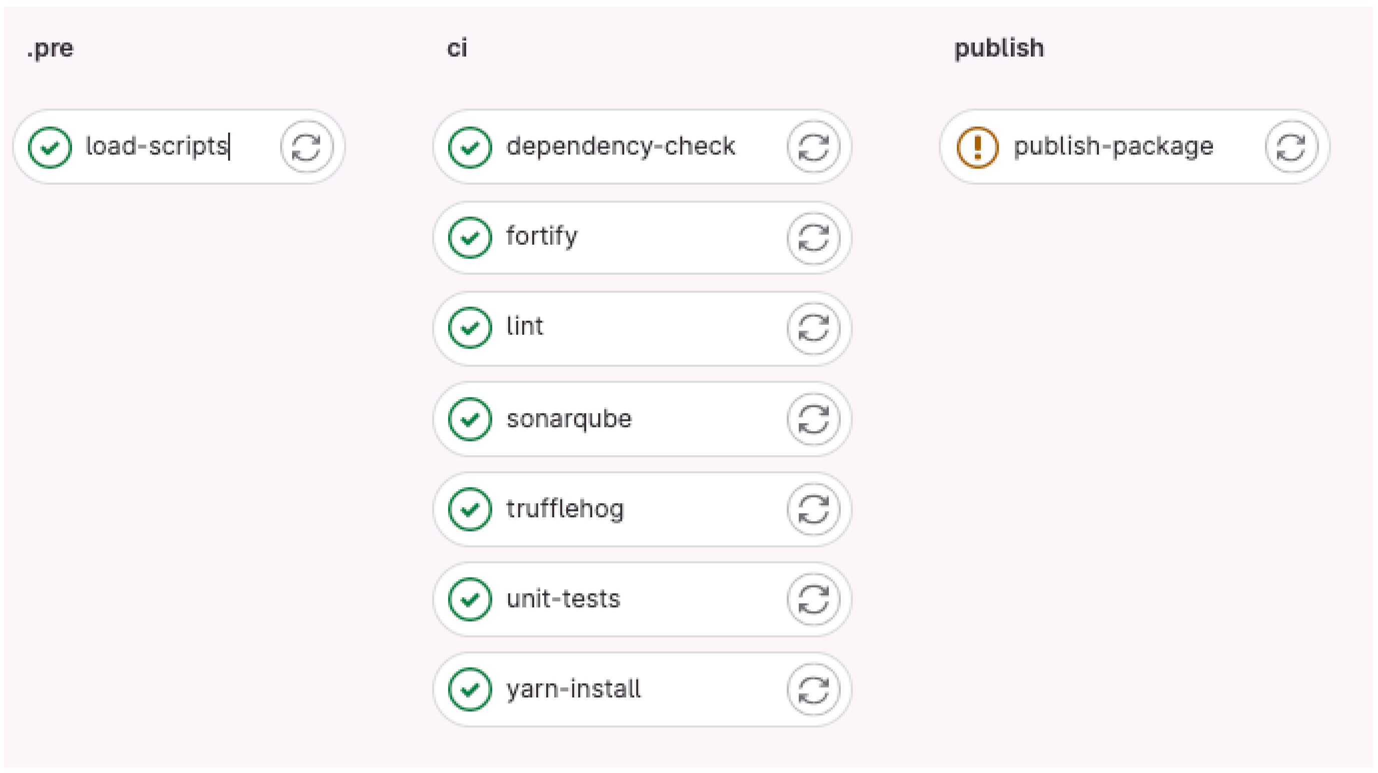
The job dependencies (NOTE: in the pipeline view, Group jobs by *job dependencies* and *show dependencies*):



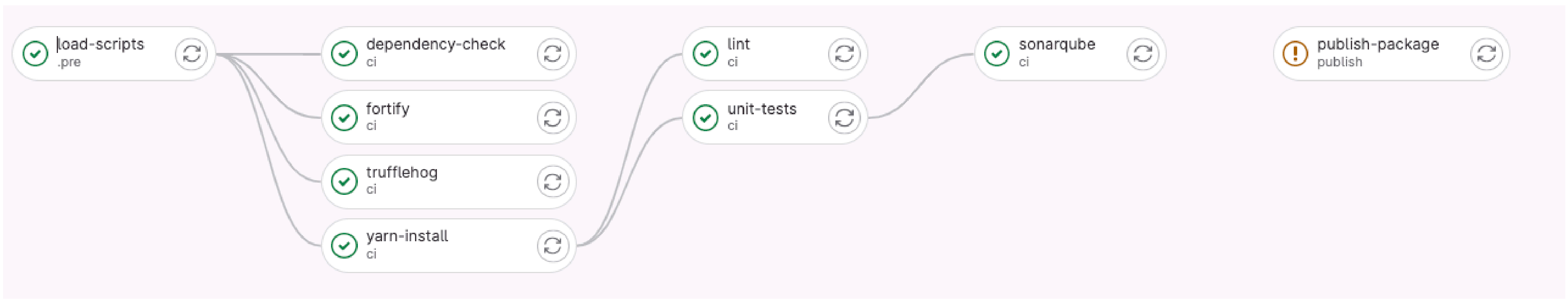
Package Pipelines

Package pipelines generate a package that will be uploaded to the project's Package Registry. It is typically used to build a common/core set of dependencies that will be used by other pipelines.

The pipeline stages and jobs:



The job dependencies (NOTE: in the pipeline view, Group jobs by *job dependencies* and *show dependencies*):



Pipeline Jobs

load-scripts

The Load Scripts job loads required scripts into the GitLab runner environment. The scripts can be found in pipeline-templates.

dockerfile-lint

The *Dockerfile Lint* job check a Dockerfile for style, format, and security issues. It will check only the Dockerfile at the root directory of the repository. This Dockerfile will be used to generate the Docker image in the *build-image* stage.

Further Details

Platform One uses a Hadolint Iron Bank image, combined with the appropriate Impact Level's container registry. Registries include the following:

- IL2 Registry
- IL4 Registry
- IL5 Registry

NOTE:
Issues labeled as **Info** severity should be addressed but will not block the pipeline; **Warnings** and **Errors** will block the pipeline.

Common Errors

Errors that occur in this job can be looked up on the Hadolint Github page, which provides links to examples of bad and fixed code.

Last user of a Dockerfile should be non-root.

Every stage of your Dockerfile should end with a non-root user. This includes intermediate stages in a multi-stage build. **USER root** can be used if you switch to another user before exiting the stage.

Example:

```
FROM image1
USER root >
COPY . .
USER node # <--- add this FROM image2 CMD [ "npm", "start" ]
```

find-authorized

The *Find Unauthorized* job will check a Dockerfile for any unauthorized commands and unauthorized registries. It will check the Dockerfile at the root directory of the repository. This Dockerfile will be used to generate the Docker image in the *build-image* stage.

Details

Only Authorized Registries

The only authorized registries for pulling images are from Ironbank (https://repo1.dso.mil) and the MDO-managed pipeline-template registries (https://registry.il2.dso.mil/, https://registry.il4.dso.mil, https://registry.il5.dso.mil). Dockerfiles pulling from other registries will fail.

Unauthorized Commands

Commands that require external web access in the Docker are not allowed. This includes commands that install dependencies. Any dependencies need to be brought in during the pipeline's *build* job – not *build-image*. The dependencies will be cached and saved as artifacts to be used throughout your pipeline. They will be available in the GitLab runner environment. They must be scanned for vulnerabilities before they are allowed to be deployed to Platform One resources.

To bring dependencies into your Docker image, use the **COPY** directive to copy files from the GitLab runner environment into the container.

Commands that update packages at the OS-level, e.g., **yum**, **dnf**, **wget**, cannot be brought in as artifacts. They must be installed on the base image. Product teams should try to find pre-approved Iron Bank images with these packages already installed. New Iron Bank images can be requested here.

We also do not allow commands that modify users.

The list of unauthorized commands can be found here: https://code.il2.dso.mil/platform-one/devops/pipeline-templates/-/blob/beta//scripts/dockerfile-scan/cmdscan/unauthorized.json

- adduser
- apk
- apt
- apt-get
- bundle add
- bundle install
- bundle update
- curl
- dnf
- dotnet publish
- g++
- gcc
- gem install
- get
- go build
- go get
- gradle build
- gradlew assemble
- grafana-cli
- install
- make
- mvn install
- mvn package

- npm ci
- npm i
- npm install
- pip install
- pipenv
- rpm
- useradd
- wget
- yarn add
- yarn install
- yum

Running Locally

Prerequisites:

- Pipeline-templates access. The *Reporter* role is sufficeint.
    - Generate a personal access token
- registry1 access
- Docker

Steps:

1. Clone pipeline-templates.

2. Run this Docker command:

```
 docker run -it --rm -w /app -v < path_to_repo >:/app -v < path_to_dir_with_pipeline-templates >/pipeline-
templates/scripts:/scripts registry1.dso.mil/ironbank/opensource/python/python39:v3.9.6 bash
```

3. Once in the Docker container, run these commands:

```
 cp /scripts/Dockerfile-scan/cmdscan/* /app cd /app python cmdscan.py Dockerfile unauthorized.json
```

Common Errors

The Dockerfile cannot be found

The Dockerfile must be at the root directory of the source code repository. It must be named Dockerfile (no affixes).

build (mvn-package, pip-install, yarn-install, gradlew-assemble, composer-install-prod, composer-install-dev, go-build, npm-ci, bundle-install, dotnet-publish, build-cpp)

Installs the dependencies for interpreted languages like JavaScript and Python through language-specific package managers (e.g., pip, npm, and bundle).

TIP:
Dependencies installed during the build job should be copied into your Dockerfile. Dependencies should not be reinstalled during the docker build.

Pipelines are configured to automatically install dependencies based on a package dependency file, (e.g., package.json, requirements.txt, and go.mod). Pipelines are designed to support one package manager. If dependencies from multiple package managers are needed, the repository needs to be broken up; package pipelines can be built to package those dependencies and upload it to the project's container registry.

Details

Please review this document for more in-depth information about build job dependencies: KB - GitLab - Build Job

Common Errors

Old dependencies or dependency versions found in project after updating the package dependency file.

The runner cache may be need to be cleared. On the pipelines page, click the **[Clear runner caches]** button.

Lint

The *Lint* job runs a linter specific to your project's language, e.g., pylint, eslint, golint. Alternate linters can be requested barring any complications in implementing them. Configuration of the linter is managed through a config file. Product teams are free to configure the rules and whitelists as desired.

Failed Job:
This job is provided as a convenience for product teams. It is not required for a CtF and a failed status will not block the pipeline.

Details

For further details, plesae refer to our HowTo - Requirements - Linting document.

unit-tests

Details

- Platform One requires 80% coverage across all code written by the team.
- A coverage file must be produced by the unit test framework (to be uploaded to SonarQube).
- For npm projects, this job is invoked by running **npm run test:unit** so you will have to define a **test:unit script** in package.json.

Common Errors

If the *Unit Test* job fails for any reason, the code coverage reporter will not be produced, and code coverage will be considered 0%.

Unit tests should not require external services (e.g., databases, authentication, and s3 buckets). For reference, refer to the following:

- https://blog.boot.dev/clean-code/writing-good-unit-tests-dont-mock-database-connections
- https://stackoverflow.com/questions/1054989/why-not-hit-the-database-inside-unit-tests

Files not written by the product team can be given exclusions.

Replicating Unit Tests Locally

Product teams may debug unit test errors in pipelines by running them inside the same image that the pipeline uses. To determine the image used by the pipeline, look at the top of the job for the following line:

```
Using Kubernetes executor with image {image name}
```

dependency-check

Dependency checks scan the dependencies for security vulnerabilities.

Who Does this?
Mission DevOps implements, but product teams fix identified security vulnerabilities.

Details

- Uses the dependencies artifact from the build job.
- Scans all dependencies against a CVE database.
- Results can be found in the SonarQube Web GUI under a separate "dependencies" project.

Common Errors

Any findings that are false positives can be reported through a Pipeline Exception Request. For each finding, justification will need to be provided as a comment. The assigned Mission DevOps engineer will review and grant the exception if the justification is valid. A single request can be ,made for multiple findings.

fortify

Fortify is a static code analyzer for Static Applications Security Testing (**SAST**), with a focus on detecting security vulnerabilities in the code.

Who Does this?
Mission DevOps implements it, the product team has full permission to mark issues and comment. Product teams are responsible for navigating to the Fortify UI, marking false-positive or won't-fix findings as "Not an Issue," then providing a comment stating why.

WARNING:
Do not suppress issues. Issues must be addressed or whitelisted by MDO. Suppressing issues will cause the pipeline job to fail.

Details

Please review this document for more information: HowTo - Requirements - Sonarqube/Fortify.

Common Errors

Any findings that are false positives can be reported through a Pipeline Exception Request. For each finding, justification will need to be provided as a comment. The assigned Mission DevOps engineer will review and grant the exception if the justification is valid. A single request can be made for multiple findings.

sonarqube

SonarQube is a static code analyzer (i.e., SAST) that checks for security vulnerabilities and general code quality issues. It will also process the code coverage report produced by the unit-tests job.

Details

Please review this document for more information: HowTo - Requirements - Sonarqube/Fortify.

Common Errors

The code coverage is checked in here, but the code coverage report is generated during the unit-test job. Verify it was generated and saved off as an artifact.

Any findings that are false positives can be reported through a Pipeline Exception Request. For each finding, justification will need to be provided as a comment. The assigned Mission DevOps engineer will review and grant the exception if the justification is valid. A single request can be made for multiple findings.

trufflehog

TruffleHog is a scanning tool to identify potential sensitive data in your repository (e.g., passwords and certificates). Product teams should use passwords for their production environment and should never be committed plaintext into your repo.

Who Does this?
Mission DevOps adds the exceptions for false positives in the product YAML.

Details

There is currently no UI for TruffleHog. Findings will be reported in the pipeline job output as well as being saved off as an artifact.

Common Errors

Refer to the troubleshooting page.

publish-package

Some pipelines are setup to publish a code artifact to GitLab instead of deploying a container to Kubernetes. We refer to them as "package pipelines." They will generate a package that will be uploaded to the GitLab package registry.

Who Does this?
Mission DevOps will create a package pipeline upon request. Versioning is generally taken from the package file used for the repository's technology, such as a package.json or pom.xml file.

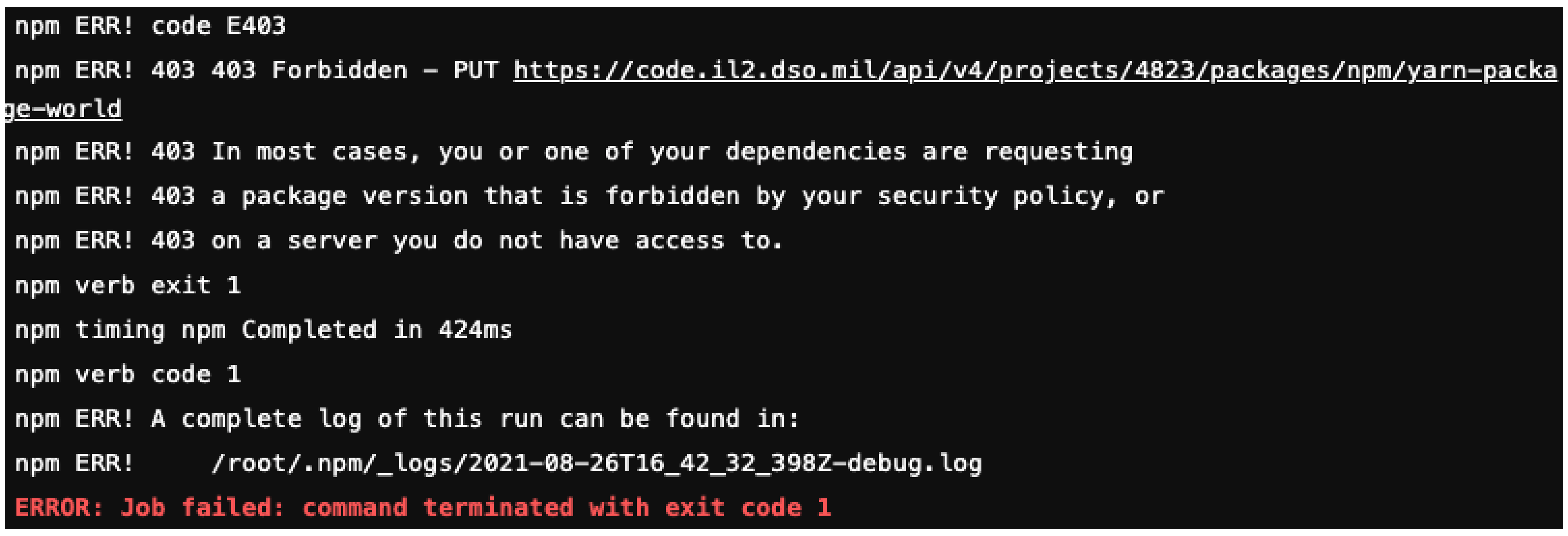
Further Details

This job runs only on the master branch.

Common Errors

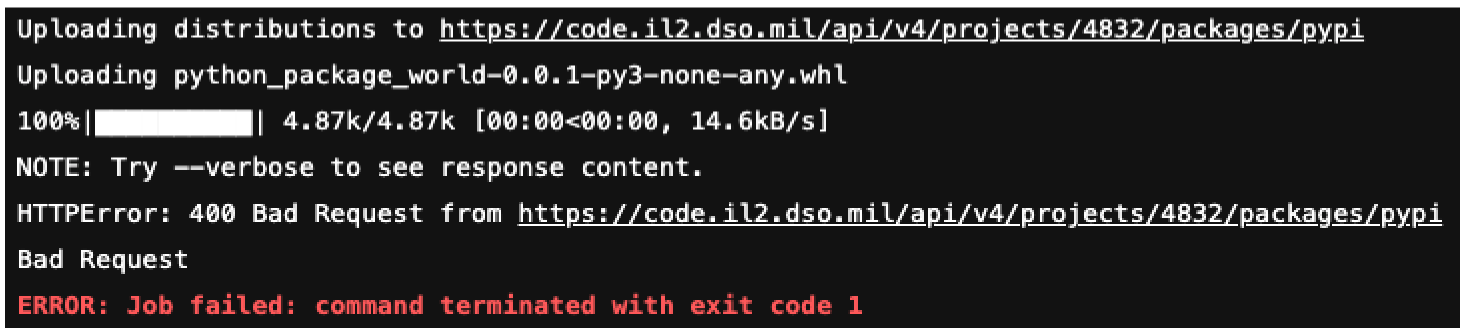
Packages must have a unique name and version. Duplicate packages will be rejected.

For npm/yarn packages, the error will look similar to:



```
npm ERR! code E403
npm ERR! 403 403 Forbidden - PUT https://code.il2.dso.mil/api/v4/projects/4823/packages/npm/yarn-packa
ge-world
npm ERR! 403 In most cases, you or one of your dependencies are requesting
npm ERR! 403 a package version that is forbidden by your security policy, or
npm ERR! 403 on a server you do not have access to.
npm verb exit 1
npm timing npm Completed in 424ms
npm verb code 1
npm ERR! A complete log of this run can be found in:
npm ERR!     /root/.npm/_logs/2021-08-26T16_42_32_398Z-debug.log
ERROR: Job failed: command terminated with exit code 1
```

For python packages, the error will look similar to:



```
Uploading distributions to https://code.il2.dso.mil/api/v4/projects/4832/packages/pypi
Uploading python_package_world-0.0.1-py3-none-any.whl
100%|██████████| 4.87k/4.87k [00:00<00:00, 14.6kB/s]
NOTE: Try --verbose to see response content.
HTTPError: 400 Bad Request from https://code.il2.dso.mil/api/v4/projects/4832/packages/pypi
Bad Request
ERROR: Job failed: command terminated with exit code 1
```

The package version must be incremented to a new unique version. Packages are uploaded to the Gitlab package registry, on the left of the repository in Gitlab click Packages & Registries → Packages.

For versioning, it is recommend to use SemVer spec.

build-image

The *build image* job builds the Docker image that will deployed. It reads the Dockerfile located at the root of the source code repository.

Further Details

This job only runs on the master branch and uses the buildah tool to generate the Docker.

twistlock-scan

This job scans the Docker image using Twistlock. It detects Common Vulnerabilities and Exposures (CVEs) as defined by the National Vulnerability Database, as well as other problems like private keys in the container.

Exceptions for CVEs in base images are granted automatically.

Details

Product Teams cannot view the Twistlock console. However, they can view Twistlock results in the pipeline build logs.

Common Errors

See common errors and solutions at: TS - Twistlock - Stage Failure.

anchore-sbom-generation

The *Anchore SBOM generation* job creates a Software Bill of Materials (SBOM), which is an artifact that provides a more complete picture of installed software. The SBOM will be used as a part of the *anchore-scan* job.

Details

This job is solely responsible on Mission DevOps. If there is a failure, report it at Pipeline Issues.

anchore-scan

The *Anchore scan* scans containers for Common Vulnerabilities and Exposures (CVEs) as defined by the National Vulnerability Database.

Details

Images must use the latest version of the base image or inherited findings will not be excluded automatically. Versions can be checked in Iron Bank Harbor.

deploy-staging

Details

Deploys the app to the Platform One staging environment.

Who Does this?
Product Team debugs errors their staging environment as reported by the ArgoCD status script.

Common Errors

< APP > FAILED to deploy in ArgoCD

Visit ArgoCD and confirm the cluster is healthy. The deploy-staging job verifies the health of the entire namespace, i.e., a failed pod for another service will cause the job to report a failure. If assistance is required, create a ticket.

e2e-tests

This job executes E2E tests.

Who Does this?
Product Team creates and fixes E2E errors.
NOTE: if your staging code is stored in one impact level and deploys to antoher, you should refer to the “Cross-IL Setups” note, provided below.

HowTo - Requirements - E2E Testing

Further Details

This job is currently designed to run Cypress tests by invoking **npm run test:e2e-ci.**

We do not currently have any requirements for what your app needs to test. See this page for more information: HowTo - Requirements - E2E Testing.

Common Errors

Errors in this job are typically due to e2e test failures. If you cannot determine the cause of the failure, please create a ticket.

pen-test

The penetration testing job is handled by OWASP ZAP.Pen test reports are uploaded to SonarQube under the *zap* project. The *zap* project will be named after the staging URL (e.g., if the staging URL is https://hello-world.staging.dso.mil, the zap project name is hello-world-staging-dso-mil-zap).

Who Does this?
The **Product Team** fixes ZAP findings.

Requirements

Requirements for this job have been documented here: HowTo-Requirements - ZAP Pen Test Job.

release-major, release minor, release patch

After a master branch passes all jobs, the **release** jobs can be run to cut a new release of the project.

Who Does this?
The **Product Team** runs this stage manually when they want to cut a release and/or promote code to production.

Details

To run the release stage, click the cog and select the type of release you want:

- **release major** will increment the major version: 1.2.3 → 2.0.0
- **release minor** will increment the minor version: 1.2.3 → 1.3.0
- **release patch** will increment the patch version: 1.2.3 → 1.2.4

When the release stage succeeds, Argo will automatically deploy the new release to prod.

Common Errors

No release for 1.X.X found.

The pipeline has not been configured to allow production deployments yet. A Certificate to field (CtF) for the app is required. If the app is ready for CtF, the process can be started here.

If a CtF has been approved and a CtF letter signed, MDO must authorize the pipeline. A request can be started here.

If the MDO request has been completed, but the error is still present; verify a new pipeline was run after the merge request for pipeline-templates was merged. Verify the release that is being cut is authorized (CtFs are valid for specific major versions).

Classified Production Deployments

Engage the ODIN team to setup your IL6/SIPR/JWICS deployments. Once engaged, the Classified Ops team will add team members to the ODIN Classified Delivery Mattermost channel.

DNS Requests

The MDO team opens a DNS Request Ticket with CNAP.

Helpful Links

- Requesting a new pipeline for staging deployment
- CtF Request
- Production Deploy Request
- Request Access to Fortify, SonarQube,and GitLab groups
- SCA Whitelisting
- General CI/CD Pipeline Assistance
- Request Access to SD Elements

- GPU Based Workloads
- Feature Request