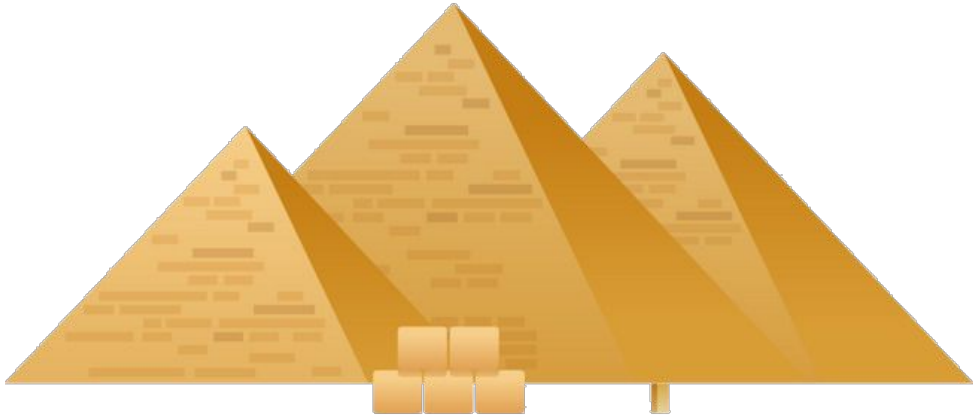


# Discover project Thoth

Thoth team - https://thoth-station.ninja

Presented by:

Maya Costantini <mcostant@redhat.com>
Fridolin Pokorny <fridolin@redthat.com>



Red Hat

# $ whoarewe

- Thoth - AIDevSecOps
  - Started (2018) as a research project in AICoE team, Office of the CTO
  - https://thoth-station.ninja
- Current members:
  - Christoph Goern, Dominik Tuchyna, Francesco Murdaca, Frido Pokorny, Gage Krumbach, Gregory Pereira, Harshad Reddy Nalla, Kevin Postlethwait, Maya Costantini, Pep Turro Mauri, Viliam Podhajecky
- See our linked YouTube channel for more information
- Follow us on Twitter - @ThothStation

# Our mission

- Help Python developers and data scientists create healthy applications

- Project has multiple parts:
  - AICoE-CI - a CI that builds container images
  - Thoth resolver - a recommendation engine for Python applications
    - AIDevSecOps
  - Dependency Monkey - a service that can validate software in a cluster
  - jupyterlab-requirements extension for managing dependencies
  - Bots maintaining GitHub repositories
  - A self hosted Python package index using Pulp available to all Red Hatters
  - Container image analysis and containerized Python applications

# Agenda

1. Introducing Python cloud based resolver
2. Benefits of resolving application dependencies in the cloud
   a. Know your runtime environment
   b. Know your Python dependencies
3. A CLI for the Python cloud based resolver
4. Demo

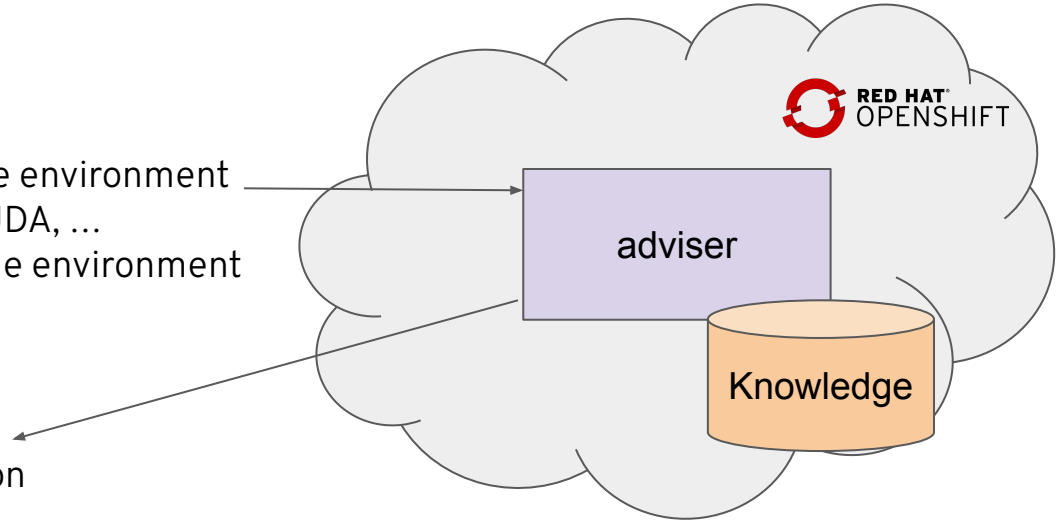
*Thoth: The cloud Python resolver*

# The Python resolver run in cloud

- Recommendation engine for Python applications
- Publicly available to the community
- Stochastic resolver implementing gradient-free reinforcement learning methods
  - Temporal difference learning is used in production
- See documentation for more information:
  - https://thoth-station.ninja/docs/developers/adviser

# Python cloud resolver

Requirements
Constraints
Information about software in runtime environment
 - OS, Python version, base image, CUDA, …
Information about hardware in runtime environment
 - CPU, GPU, …
Static source-code analysis
Recommendation type




Lockfile + justification

```
$ pip install thamos
$ thamos config
$ thamos advise
```

*Declarative interface for the resolver to resolve Python packages following prescribed rules*

# Prescriptions - declarative interface to the cloud based resolver

- Provide a way to declaratively state how the resolution process should look like
- Community driven open database used by the resolver to resolve high quality software - you can contribute!
  - https://github.com/thoth-station/prescriptions/
- A set of YAML files that are automatically consumed by the resolver in a deployment
- See documentation for more information:
  - https://thoth-station.ninja/docs/developers/adviser/prescription.html

# Prescriptions - Example

- Pillow in version 8.3.0 does not work with NumPy

  https://github.com/python-pillow/Pillow/issues/5571

```
with PIL.Image.open(filepath) as img:
      numpy.array(img, dtype=numpy.float32)
```

```
>          frame_paletted = np.array(im, np.uint8)
E          TypeError: __array__() takes 1 positional argument but 2 were given

/lib/python3.9/site-packages/imageio/plugins/pillow.py:745: TypeError
```

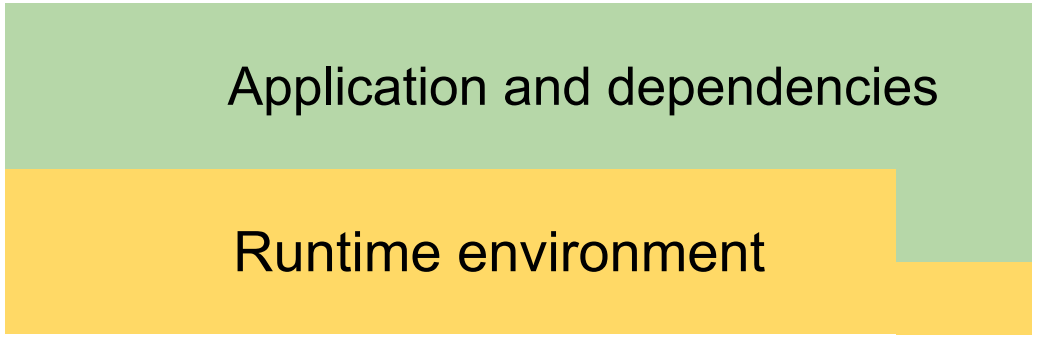


Red Hat

```yaml
units:
 steps:
 - name: Pillow830TypeErrorStep
   type: step
   should_include:
     adviser_pipeline: true
   match:
     package_version:
       name: pillow
       version: ==8.3.0
       index_url: https://pypi.org/simple
     state:
       resolved_dependencies:
       - name: numpy
   run:
     not_acceptable: Pillow in version 8.3.0 does not work with NumPy
     stack_info:
     - type: WARNING
       message: Pillow in version 8.3.0 does not work with NumPy
       link: https://github.com/python-pillow/Pillow/issues/5571
```

# Fitting the runtime environment

- Checking RPM, Python packages, ABI, CUDA, cuDNN, OpenMKL, ...
- One central place to declare requirements
  - Prescriptions (packages)
  - User's runtime environment (Pipfile, .thoth.yaml file)
- Still optional, the resolution might be "generic", like pip, Pipenv, ...






Red Hat

# Security - AIDevSecOps

- Docs: Thoth security advises
- Recommendations based on static source code analysis
  - See recommendations from the Python standard library (example)
- PyPA - advisory-db
  - A database of known vulnerabilities in Python ecosystem
  - https://github.com/pypa/advisory-db
- Security Scorecards by Open Source Security Foundation
  - https://openssf.org/blog/2020/11/06/security-scorecards-for-open-source-projects/
  - Example: see scorecards_prefixed prescriptions for TensorFlow
- + additional information about Python packages not strictly related to security

# 🖥️ Demo

```
pip install thamos
thamos config
thamos add “flask~=0.12”
thamos advise
```

Look at the tutorial documentation to reproduce the demo

https://redhat-scholars.github.io/managing-vulnerabilities-with-thoth

# Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat



Red Hat

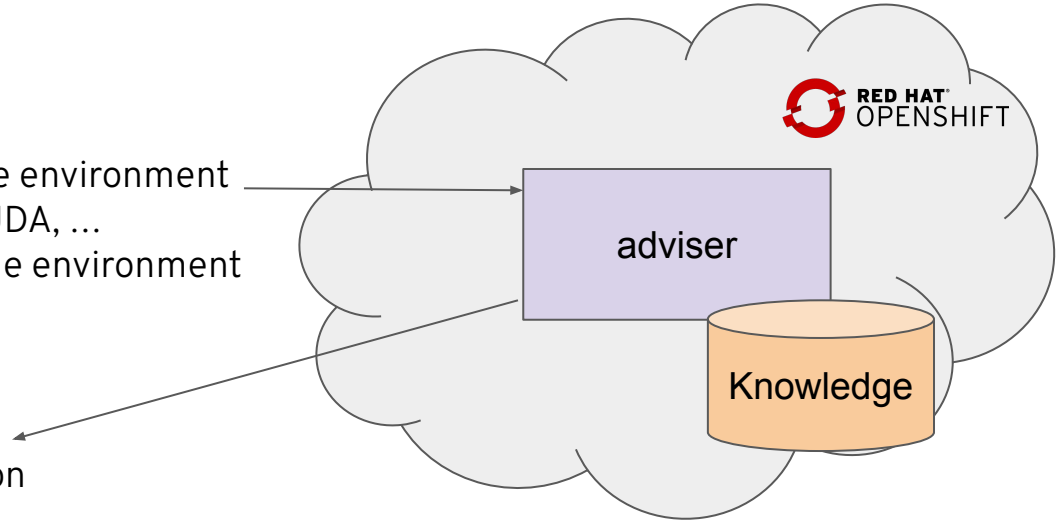
*Thoth: The cloud Python resolver*

# The Python resolver run in cloud

- Recommendation engine for Python applications
- Publicly available to the community
- Stochastic resolver implementing gradient-free reinforcement learning methods
  - Temporal difference learning is used in production
- See documentation for more information:
  - https://thoth-station.ninja/docs/developers/adviser

# Python cloud resolver

Requirements
Constraints
Information about software in runtime environment
 - OS, Python version, base image, CUDA, …
Information about hardware in runtime environment
 - CPU, GPU, …
Static source-code analysis
Recommendation type




Lockfile + justification

```
$ pip install thamos
$ thamos config
$ thamos advise
```

*Demo*

# Runtime environments and recommendation types

- Concept of "*overlays*"
  - Each overlay states requirements for the application for the given runtime environment
  - Ex. runtime environment for inference and runtime environment for training
    - Can have different requirements and runtime environments

- Support for "*recommendation types*"
  - Different resolution considering intention with the application
  - https://thoth-station.ninja/recommendation-types/
  - Ex. production environment should be secure, isolated environment where the training is done should be performant



Red Hat

```yaml
host: khemenu.thoth-station.ninja
requirements_format: pipenv

runtime_environments:
- name: "inference"
  recommendation_type: security
  operating_system:
    name: "rhel"
    version: "8"
  python_version: "3.8"

- name: "training"
  recommendation_type: performance
  operating_system:
    name: "rhel"
    version: "8"
  cuda_version: "11.1"
  python_version: "3.8"
```

```toml
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi-org-simple"

[packages]
flask = "*"
tensorflow = "~=2.4"

[dev-packages]

[requires]
python_version = "3.8"

[thoth]
disable_index_adjustment = false

[thoth.allow_prereleases]
```

```toml
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi-org-simple"

[[source]]
url = "https://thoth-station.ninja/simple"
verify_ssl = true
name = "aicoe"


[packages]
boto3 = "*"
tensorflow = {"version"="~=2.4", "index"="aicoe"}

[dev-packages]

[requires]
python_version = "3.8"

[thoth]
disable_index_adjustment = true

[thoth.allow_prereleases]
protobuf = true
```

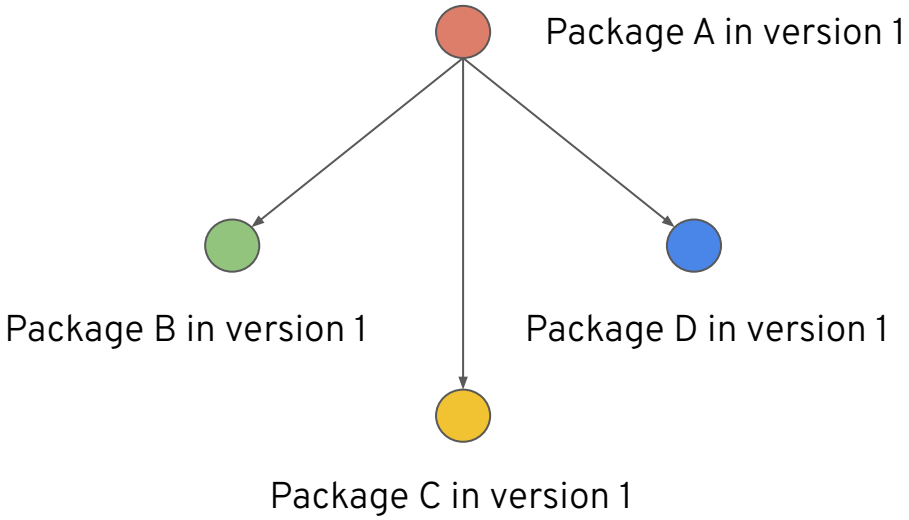
🕒 recommendations can take some time (now: 20mins)

# Why gradient-free reinforcement learning?

- Wide range (*approx. infinite?*) of possible resolutions depending on requirements used in the application and other inputs to the resolver
- The resolution process is seen as a Markov decision process
- A model is trained on each request to the resolver
- Exploration phase (*stochastic*) and subsequent exploitation phase comes with the resolved software stack meeting desired criteria

# Why reinforcement learning?

- Observations on shared sub-graphs

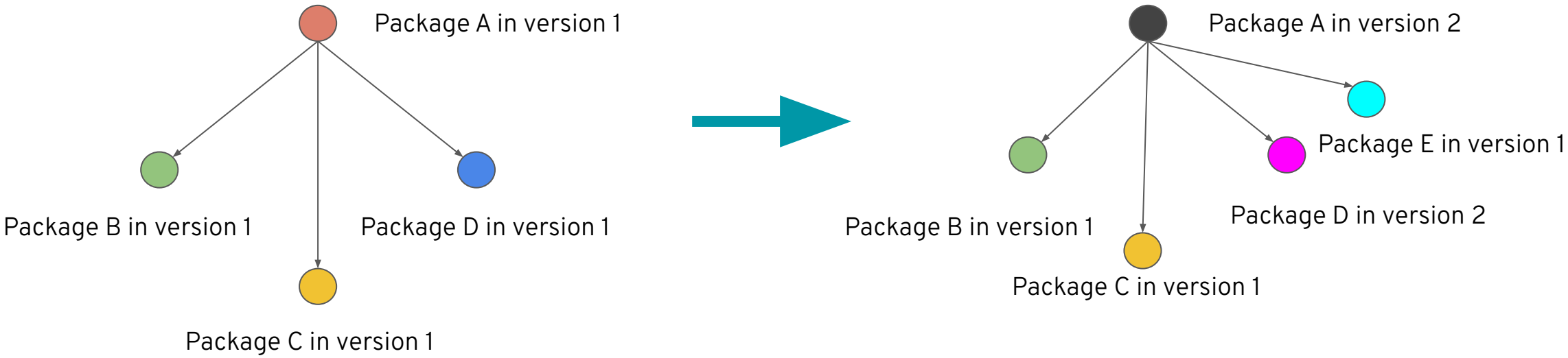





Red Hat

# Why reinforcement learning?

- Observations on shared sub-graphs

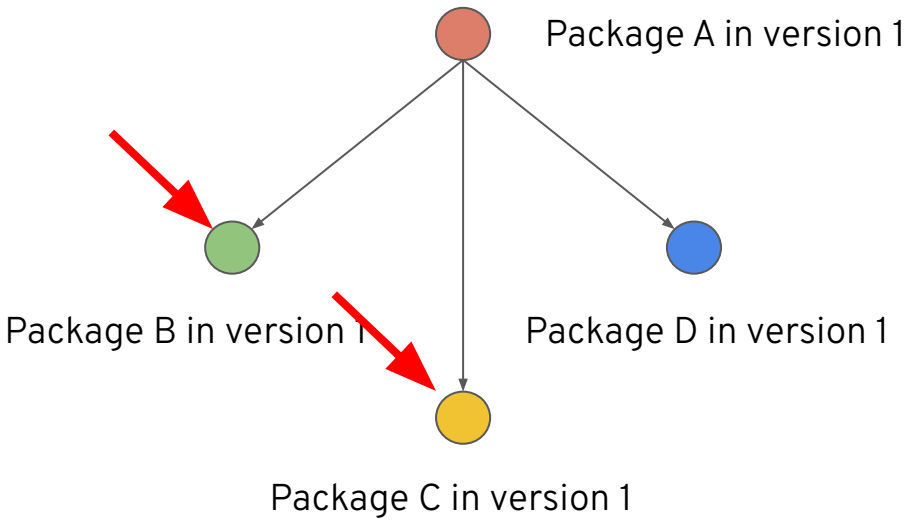


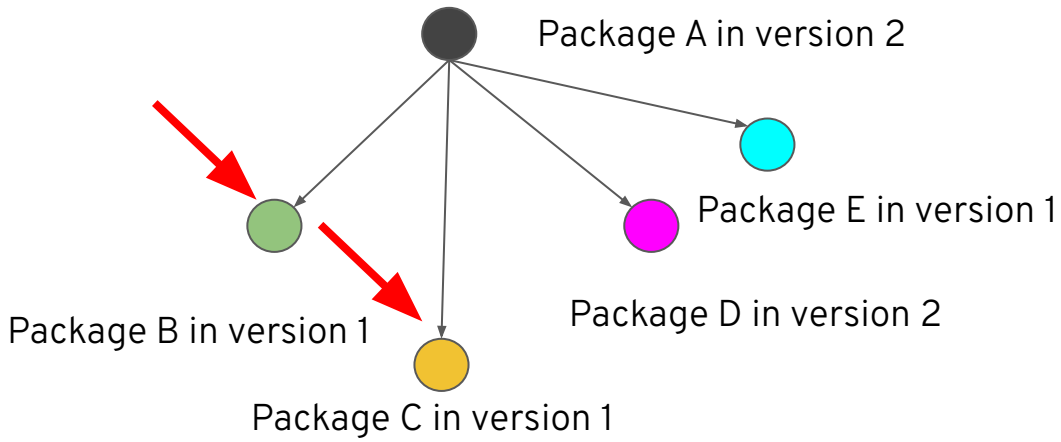



Red Hat

# Why reinforcement learning?

- Observations on shared sub-graphs






Red Hat

# Resolution pipeline

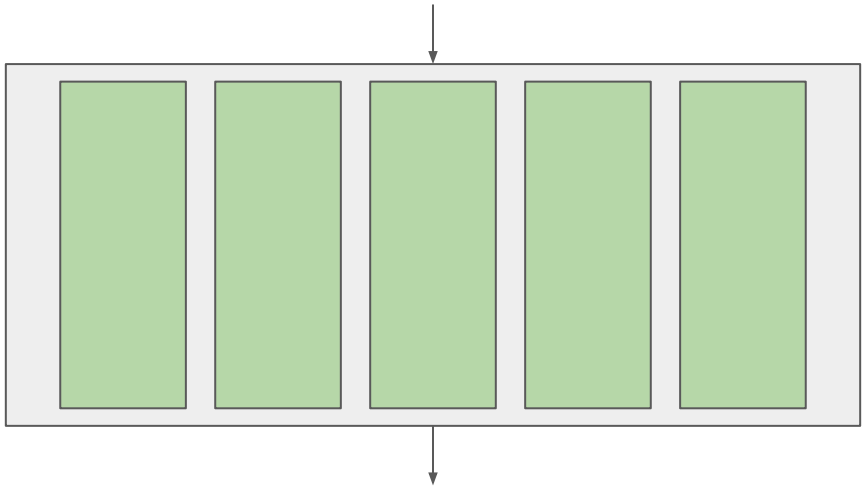
- The resolution process is using a pipeline made out of units of different types
  - Base pipeline types: boots, pseudonyms, sieves, steps, strides, wraps
  - https://thoth-station.ninja/docs/developers/adviser/pipeline.html

- Pipeline units can be implemented directly in Python or declarativelly in YAML files

- The resolution pipeline is constructed dynamically based on inputs to the resolution engine

# Resolution pipeline

Requirements
Constraints
Information about software in runtime environment
 - OS, Python version, base image, CUDA, …
Information about hardware in runtime environment
 - CPU, GPU, ...
Static source-code analysis
Recommendation type



Lockfile + justification

# Resolution pipeline

- Operations on top of dependency graph
  - Fix overpinning issues
  - Fix underpinning issues
  - Adding new dependencies and whole dependency sub-graphs
  - Removing dependencies and the whole dependency sub-graphs
  - Adding “pseudonyms” - tensorflow Vs. intel-tensorflow (different builds)
  - Scoring actions taken during the resolution process
- Additional information to users - stack level guidenenance or guidance on the runtime environments used
- Guidenance based on static source code analysis

*Declarative interface for the resolver to resolve Python packages following prescribed rules*

# Prescriptions - declarative interface to the cloud based resolver

- Provide a way to declarativelly state how the resolution process should look like
- Community driven open database used by the resolver to resolve high quality software
  - https://github.com/thoth-station/prescriptions/
- A set of YAML files that are automatically consumed by the resolver in a deployment
- See documentation for more information:
  - https://thoth-station.ninja/docs/developers/adviser/prescription.html

# Prescriptions

*When using OpenShift or Kubernetes, one provides manifest files that state how the desired state of a cluster should look like. Prescriptions might be seen analogous to this - prescriptions provide a way to declaratively state how the desired dependency resolution should look like considering the prescribed rules. Then, it's up to the reinforcement learning algorithm implemented in Thoth's adviser to find a solution in the form of a lockfile respecting the prescribed rules, requirements for the application and other inputs to the Thoth's cloud resolver.*

# Prescriptions - Example 1

- Pillow in version 8.3.0 does not work with NumPy

  https://github.com/python-pillow/Pillow/issues/5571

```
with PIL.Image.open(filepath) as img:
      numpy.array(img, dtype=numpy.float32)
```

```
>          frame_paletted = np.array(im, np.uint8)
E          TypeError: __array__() takes 1 positional argument but 2 were given

/lib/python3.9/site-packages/imageio/plugins/pillow.py:745: TypeError
```



Red Hat

```yaml
units:
 steps:
 - name: Pillow830TypeErrorStep
   type: step
   should_include:
     adviser_pipeline: true
   match:
     package_version:
       name: pillow
       version: ==8.3.0
       index_url: https://pypi.org/simple
     state:
       resolved_dependencies:
       - name: numpy
   run:
     not_acceptable: Pillow in version 8.3.0 does not work with NumPy
     stack_info:
     - type: WARNING
       message: Pillow in version 8.3.0 does not work with NumPy
       link: https://github.com/python-pillow/Pillow/issues/5571
```




Red Hat

```yaml
units:
steps:
- name: Pillow830TypeErrorStep
  type: step
  should_include:
    adviser_pipeline: true
  match:
    package_version:
      name: pillow
      version: ==8.3.0
      index_url: https://pypi.org/simple
    state:
      resolved_dependencies:
      - name: numpy
  run:
    not_acceptable: Pillow in version 8.3.0 does not work with NumPy
    stack_info:
    - type: WARNING
      message: Pillow in version 8.3.0 does not work with NumPy
      link: https://github.com/python-pillow/Pillow/issues/5571
```

```yaml
units:
 steps:
 - name: Pillow830TypeErrorStep
   type: step
   should_include:
     adviser_pipeline: true
   match:
     package_version:
       name: pillow
       version: ==8.3.0
       index_url: https://pypi.org/simple
     state:
       resolved_dependencies:
       - name: numpy
   run:
     not_acceptable: Pillow in version 8.3.0 does not work with NumPy
     stack_info:
     - type: WARNING
       message: Pillow in version 8.3.0 does not work with NumPy
       link: https://github.com/python-pillow/Pillow/issues/5571
```

```yaml
units:
 steps:
 - name: Pillow830TypeErrorStep
  type: step
  should_include:
   adviser_pipeline: true
  match:
   package_version:
    name: pillow
    version: ==8.3.0
    index_url: https://pypi.org/simple
   state:
    resolved_dependencies:
    - name: numpy
  run:
   not_acceptable: Pillow in version 8.3.0 does not work with NumPy
   stack_info:
   - type: WARNING
    message: Pillow in version 8.3.0 does not work with NumPy
    link: https://github.com/python-pillow/Pillow/issues/5571
```

```yaml
units:
 steps:
 - name: Pillow830TypeErrorStep
   type: step
   should_include:
     adviser_pipeline: true
   match:
     package_version:
       name: pillow
       version: ==8.3.0
       index_url: https://pypi.org/simple
     state:
       resolved_dependencies:
       - name: numpy
   run:
     not_acceptable: Pillow in version 8.3.0 does not work with NumPy
     stack_info:
     - type: WARNING
       message: Pillow in version 8.3.0 does not work with NumPy
       link: https://github.com/python-pillow/Pillow/issues/5571
```

```yaml
units:
 steps:
 - name: Pillow830TypeErrorStep
   type: step
   should_include:
     adviser_pipeline: true
   match:
     package_version:
       name: pillow
       version: ==8.3.0
       index_url: https://pypi.org/simple
     state:
       resolved_dependencies:
       - name: numpy
 run:
   not_acceptable: Pillow in version 8.3.0 does not work with NumPy
   stack_info:
   - type: WARNING
     message: Pillow in version 8.3.0 does not work with NumPy
     link: https://github.com/python-pillow/Pillow/issues/5571
```




Red Hat

# Prescriptions - Example 2

*Adjust requirements in GPU enabled environments.*

- Use tensorflow-gpu as a "*pseudonym*" to tensorflow if GPU enabled environment is available
  - tf_gpu.yaml
- Use the right tensorflow-gpu for the environment following TF support matrix
  - https://www.tensorflow.org/install/source#gpu
  - tf_cuda.yaml
  - tf_cudnn.yaml

# Prescriptions - Example 3

*Fixing library overpinning issues.*

- TensorFlow in version 2.1 can cause runtime errors when running with h5py>=3 caused by overpinning
  - tf_21_h5py.yaml
  - https://thoth-station.ninja/j/tf_21_h5py.html

# Prescriptions - Example 4

*Use a specific Python package index providing optimized wheel builds.*

- Prioritize resolving AICoE builds of TensorFlow for AVX2 enabled environments
  - tf_avx2.yaml
  - https://thoth-station.ninja/j/aicoe_tf_avx2.html
- Use *only* CUDA 11.1 builds of torch available on a PyTorch index:
  - gpu_index.yaml

# Prescriptions - Example 5

*Block using certain library functions due to security reasons.*

- mktemp is deprecated due to vulnerability to race conditions
  - tempfile.yaml

# Prescriptions - Example 6

*Check configuration of the runtime environment. Pipeline units can also act on the base image used.*

- A GPU is available but no CUDA is available
  - gpu_no_cuda.yaml

# Prescriptions - Example 7

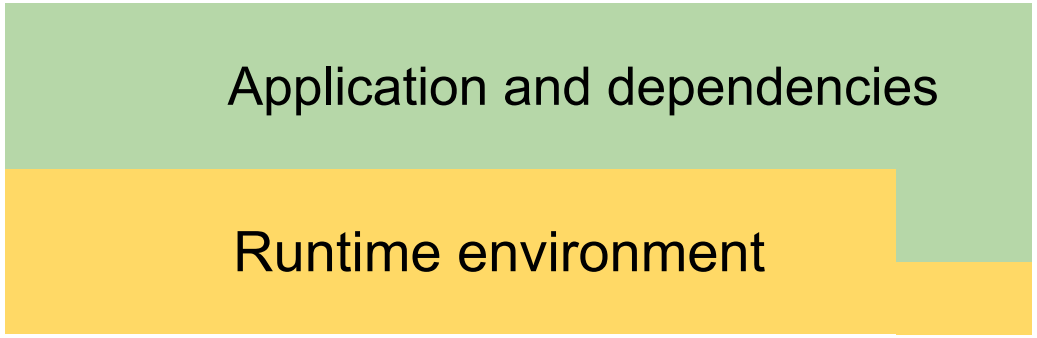
*Resolve considering RPM packages available in the runtime environment (similarly ABI or Python packages available).*

- GitPython requires Git present in the runtime environment
  - Rpm.yaml

- Python is a dynamic programming language
  - Foreign library function calls on runtime (ctypes) - `cdll.LoadLibrary("libc.so.6")`
  - Manylinux standards declaring runtime requirements are slow, no other option
  - => declare requirement for the package using prescriptions

# Fitting the runtime environment

- Checking RPM, Python packages, ABI, CUDA, cuDNN, OpenMKL, ...
- One central place to declare requirements
  - Prescriptions (packages)
  - User's runtime environment (Pipfile, .thoth.yaml file)
- Still optional, the resolution might be "generic", like pip, Pipenv, ...

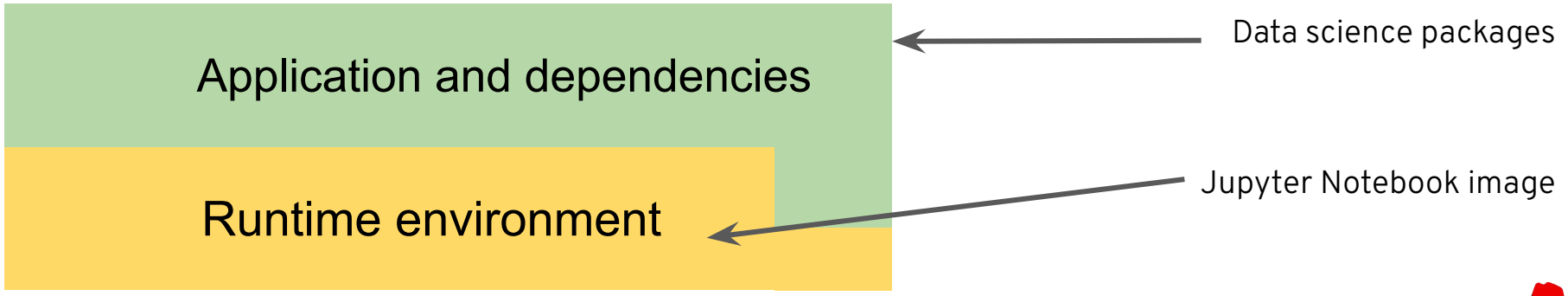





Red Hat

# Fitting the runtime environment

- Checking RPM, Python packages, ABI, CUDA, cuDNN, OpenMKL, ...
- One central place to declare requirements
  - Prescriptions (packages)
  - User's runtime environment (Pipfile, .thoth.yaml file)
- Still optional, the resolution might be "generic", like pip, Pipenv, ...

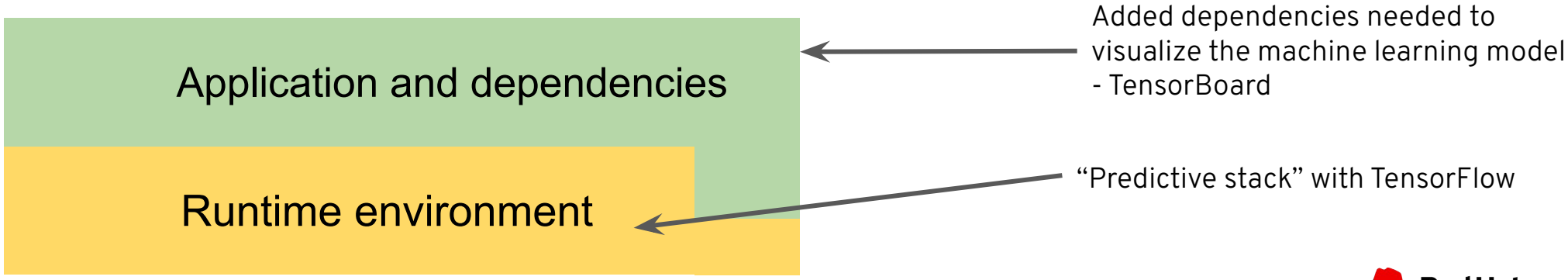





Red Hat

# Fitting the runtime environment

- Checking RPM, Python packages, ABI, CUDA, cuDNN, OpenMKL, ...
- One central place to declare requirements
  - Prescriptions (packages)
  - User's runtime environment (Pipfile, .thoth.yaml file)
- Still optional, the resolution might be "generic", like pip, Pipenv, ...






Red Hat

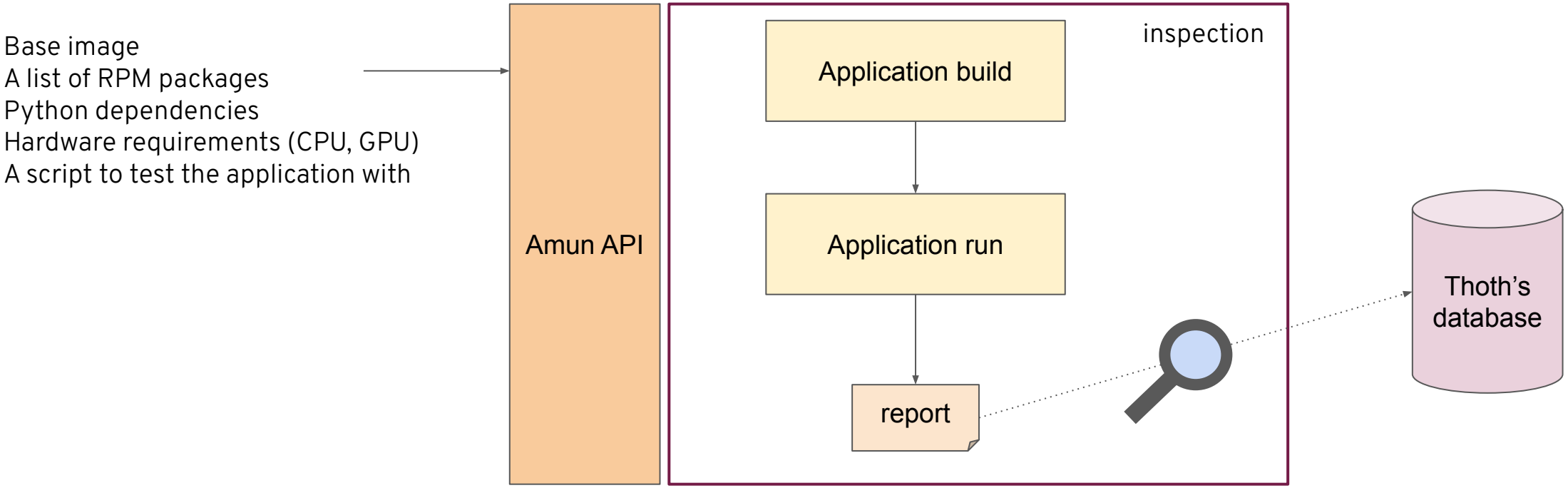
*Dependency Monkey*

# Dependency Monkey

- A service capable of evaluating different combinations of Python packages that can be resolved considering dependency graph
- Resolve a valid resolution which is then tested in the cluster
  - Resolution is offline using thoth-adviser, unlike pip
  - The resolved packages are tested and the knowledge is derived
    - Specifically to runtime environment (OS, Python version, ...)
    - Specifically to hardware available in the cluster
- Documentation available online
  - https://thoth-station.ninja/docs/developers/adviser/dependency_monkey.html

# Amun API

- A service that can test the application with the resolved stack
  - https://github.com/thoth-station/amun-api






Red Hat

# Observed issues

- Dependency Monkey Zoo
  - https://github.com/thoth-station/dependency-monkey-zoo
  - Inspections we run to verify software quality

- AI software stack inspection with Thoth and TensorFlow
  - https://thoth-station.ninja/j/tf_21_urllib3.html
  - Prescription to fix the issue: tf_21_urllib3.yaml

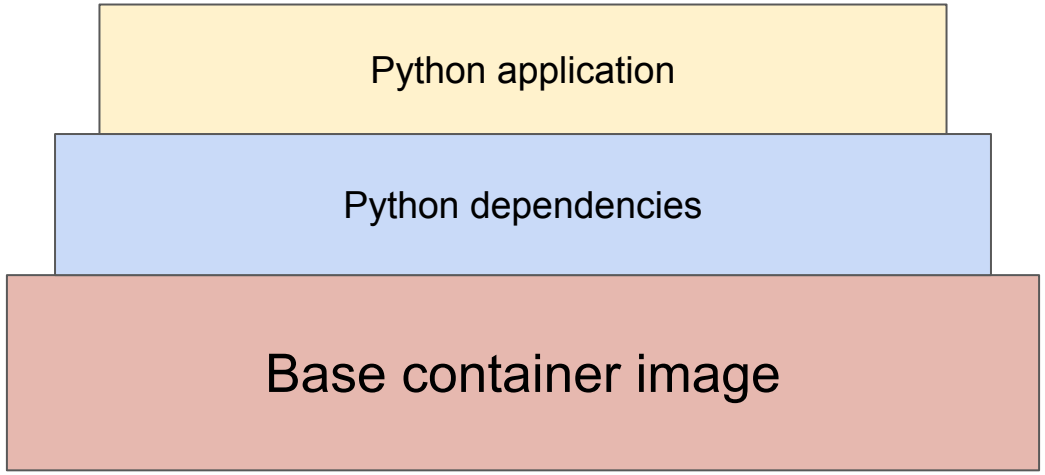
*Base container images and their analyses*

# Base container images and their analyses

- Thoth team provides container images to run the Python applications
  - Runtime environments (S2I): ubi8-py38, f34-py39, ...
    - https://github.com/thoth-station/s2i-thoth
  - Jupyter Notebooks, torch container images, ...
    - https://github.com/thoth-station/ps-cv
- These container images are automatically analyzed once built
  - Extracting content (RPM packages, ABI available, Python packages, ...)
- Thoth provides an endpoint for container image analysis also for users

# Base container images

- Provide base for running the application
- Python application stack is installed into the base container image
- The recommendation engine can take into account content already shipped in the base container image (ABI, Python applications, RPM packages, ...)






Red Hat

# Build log analyses

- Analyze build logs produced during container image build
- micropipenv - now part of vanilla Python s2i
  - https://github.com/thoth-station/micropipenv/
  - micropipenv: Installing Python dependencies in containerized applications
- Logs are automatically aggregated and stored in Thoth
  - What was the issue? Can we help by adjusting the resolution, base image, ...
  - Automatic parsing of logs into JSON (automated extraction of errors, packages installed,...)
  - AICoE-CI integration - https://github.com/AICoE/aicoe-ci
  - OpenShift builds: build-watcher - https://github.com/thoth-station/build-watcher

*Security - AIDevSecOps*



Red Hat

# Security - AIDevSecOps

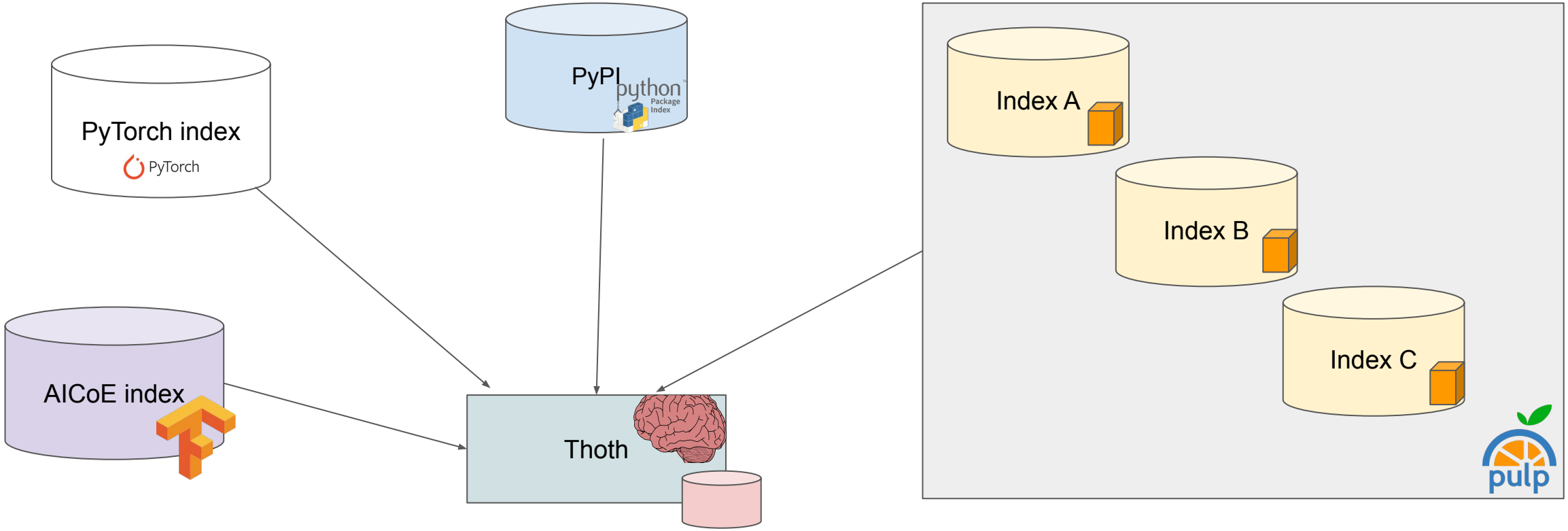
- Docs: Thoth security advises
- Recommendations based on static source code analysis
  - See recommendations from the Python standard library (example)
- PyPA - advisory-db
  - A database of known vulnerabilities in Python ecosystem
  - https://github.com/pypa/advisory-db
- Security Scorecards by Open Source Security Foundation
  - https://openssf.org/blog/2020/11/06/security-scorecards-for-open-source-projects/
  - Example: see scorecards_prefixed prescriptions for TensorFlow
- Bandit - static source code analysis run in the cluster
  - https://github.com/PyCQA/bandit
  - Security indicators

*Python package index hosted by Red Hat*

# Pulp Python Package Index

- Established a recurrent meeting to gather requirements from teams using Python at Red Hat
  - 🐍 Python package index hosted by Red Hat
- Pulp deployed with pulp_python plugin on Operate First
  - https://www.operate-first.cloud/users/pulp/usage.md
- Soon to be available for use, cluster migration postponed the release

# Python package indexes & Thoth






Red Hat

# Python package indexes

- Automatically monitored in a Thoth deployment
  - builds outside of manylinux standards:
    - index with CUDA specific builds
    - AVX2 optimized builds of TensorFlow
- Packages published on monitored Python package indexes are automatically analyzed
- The set of packages to be analyzed is configurable (name, version range, index)
- Cross index resolution without dependency confusion
- One central point of knowledge in cloud based resolver
- Releasing to a Python package index - AICoE-CI
  - https://github.com/thoth-station/aicoe-ci-pulp-upload-example

*Thoth's technology stack*

# Technology stack

- OpenShift
- Python, some resolver parts optimized in C and C++
- Ceph
- PostgreSQL
- Prometheus, Grafana Dashboards
- Kafka
- Argo Workflows
- Argo CD
- Sentry
- Tekton pipelines (AICoE-CI)