Thanatos

Codename: Thanatos Θ

Pat McLean

psmware ltd

Table of contents

1. Thanatos Θ

1.1 Introduction

Codename: Thanatos

Last update: May 22, 2021

2. Getting Involved

2.1 Thanatos Development

Thanatos is maintained as a GitHub project under the GNU Affero General Public License 3 license. Users are encouraged to submit GitHub issues for feature requests and bug reports.

2.1.1 Governance

Thanatos is a community-based Free Open Source Software (FOSS) project sponsored by Virtual Service Operations, LLC (VSO). As a solution provider, VSO works with its clients around the world to craft and build Service Management, Migration and Managed Services of Hybrid Cloud Environments strategies and solutions. The direction of this project will be shaped by VSO, as well as by input from VSO customers; independent of where requests come from, contributors will need to follow the Contributing Guidelines.

The Thanatos Core Team is responsible for the direction and execution of the code that gets committed to the project.

The following individuals are on the Thanatos Core Team:

- Laura Richardson
- Patrick McLean
- Thom Walters

2.1.2 Contributing

We welcome many forms of contributions to Thanatos. While we understand most contributions will commonly come from developers, we encourage others to contribute in the form of docs, tutorials, and user guides. If you have other ideas for contributing, don't hesitate to open an issue or have a discussion in one of the forums below.

Release Management

In order to best understand how to contribute and where to open an issue or discussion, you should understand how work moves from idea to feature and how the roadmap is structured.

There are three major "buckets" of work to be aware of within the lifecycle of getting contributions committed and released:

- **Current** - Work that is planned for the release currently being developed.
- **Near Term** - Work that is planned for one of the next two releases after the one currently being developed.
- **Future** - Work that needs more discussion and/or will be planned for a version three or more releases later.

The following provides more detail on these.

CURRENT

- Current tickets (GitHub issues) that are being worked on for the *current* release or bugs that are found and will be fixed in the *current* release.
- Uses `current` label on GitHub.
- The GitHub **Release Milestone** will track items for the *current* release.

> **Note**
>
> Release windows and dates will be updated per Release Management.

NEAR TERM

- Current tickets (GitHub issues) that are estimated to complete in one of the next two releases, e.g. 3-6 months to get into core, if accepted.
- GitHub discussions are used to create one or more GitHub issues when and if something moves from *Future* to *Near Term*.
- Uses `near-term` label on GitHub.

FUTURE

- Work that is for 3+ releases away or work that needs more free form discussions and brainstorming to better scope future bodies of work.
- Estimated 7+ months to get into core, if accepted.
- GitHub Discussions are used for collaborating on *future* work.
- If a GitHub issue is opened and is deemed that it is out of scope for *Current* or *Near Term*, it will be converted into a GitHub Discussion.
- GitHub Discussions will be closed when the topic/feature moves from *Future* to *Near Term*.

Over time, the process of moving work from *Future* to *Near Term* to *Current* will continue to get further refined.

Release Schedule

Here is what you need to know about Thanatos releases:

- The core team estimates quarterly releases with the majority of them being minor releases.
- It is an aspirational goal that there will be no more than one major release per year as major releases do indicate a break in backwards compatibility.
- Patch releases will be released as needed without a defined schedule.
- Patch releases will be used for bugs, security vulnerabilities, backports, and other issues as they arise.
- Given the core team is estimating quarterly releases, there will not be firm dates for releases.
- In order to provide more visibility into the development and release schedule of Thanatos, there will be structured notifications as follows:
- At the start of a release cycle, the estimated timeframe for release will be a 4-6 week window.
- Halfway through the release cycle (~6 weeks), the estimated timeframe for release will be narrowed to a 3-4 week window
- After 8-9 weeks within the development cycle, the estimated timeframe for release will be narrowed further to a 2 week window.
- The final notification will be provided 3-5 days before the release drops.
- The dates and notifications will occur by updating the GitHub Release Milestone and on Slack.

For 2021, the team estimates there will be three more releases with no more than one of them being a major release.

Long Term Support (LTS)

It is the core team's intention to have a Long Term Support (LTS) version of Thanatos. The initial target release for the LTS version is the end of 2021, which will be the third or fourth release of Thanatos. Being that Thanatos is a new and open source community-based project, the goal is to collect as much feedback as possible within the first 3-6 months that will help finalize the correct LTS model.

Deprecation Policy

The deprecation policy will be such that there will be at least one release that makes users aware of a feature that will be deprecated in the next release.

Versioning

Semantic Versioning (SemVer) is used for Thanatos versioning.

Contributor Workflow

The following documents the lifecycle of work within Thanatos.

1. Open/request a feature enhancement or file a bug a. If bug, see here b. If feature request or enhancement, continue.
2. Open a GitHub Issue a. The issue will be reviewed. Based on the request, it will get labeled as `current`, `near-term`, `future`. b. It will likely only stay in *current* if it is trivial and quick work. c. If it gets labeled as *future*, the issue will be closed in the next batch of issues that get migrated and converted to GitHub discussions.

For any issue that receives a label of `current` or `near-term`, it will also receive a label of `status: accepted` or `status: blocked`.

If you follow these steps, there **will** be a GitHub Issue opened prior to submitting a Pull Request (PR). However, we're quite aware that a PR may come in without ever being discussed in an Issue or Discussion. While we do not advocate for this, you should be aware of the process that will be followed for those circumstances.

Should this happen and if you followed the project guidelines, have ample tests, code quality, you will first be acknowledged for your work. So, thank you in advance! After that, the PR will be quickly reviewed to ensure that it makes sense as a contribution to the project, and to gauge the work effort or issues with merging into *current*. If the effort required by the core team isn't trivial, it'll likely still be a few weeks before it gets thoroughly reviewed and merged, thus it won't be uncommon to move it to *near term* with a `near-term` label. It will just depend on the current backlog.

Communication

Communication among the contributors should always occur via public channels. The following outlines the best ways to communicate and engage on all things Thanatos.

GITHUB

- GitHub issues - All feature requests, bug reports, and other substantial changes should be documented in an issue.
- GitHub discussions - The preferred forum for general discussion and support issues. Ideal for shaping a feature request prior to submitting an issue.

GitHub's discussions are the best place to get help or propose rough ideas for new functionality. Their integration with GitHub allows for easily cross-referencing and converting posts to issues as needed. There are several categories for discussions:

- **General** - General community discussion.
- **Ideas** - Ideas for new functionality that isn't yet ready for a formal feature request. These ideas are what will be in scope to review when moving work from *Future* to *Near Term* as stated in the previous section.
- **Q&A** - Request help with installing or using Thanatos.

Contributing to Thanatos

REPORTING BUGS

- First, ensure that you're running the latest stable version of Thanatos. If you're running an older version, it's possible that the bug has already been fixed.
- Next, check the GitHub issues list to see if the bug you've found has already been reported. If you think you may be experiencing a reported issue that hasn't already been resolved, please click "add a reaction" in the top right corner of the issue and add a thumbs up (+1). You might also want to add a comment describing how it's affecting your installation. This will allow us to prioritize bugs based on how many users are affected.
- When submitting an issue, please be as descriptive as possible. Be sure to provide all information request in the issue template, including:
- The environment in which Thanatos is running
- The exact steps that can be taken to reproduce the issue

- Expected and observed behavior
- Any error messages generated
- Screenshots (if applicable)
- Please avoid prepending any sort of tag (e.g. "[Bug]") to the issue title. The issue will be reviewed by a maintainer after submission and the appropriate labels will be applied for categorization.
- Keep in mind that bugs are prioritized based on their severity and how much work is required to resolve them. It may take some time for someone to address your issue.

FEATURE REQUESTS

- First, check the GitHub issues list and Discussions to see if the feature you're requesting is already listed. (Be sure to search closed issues as well, since some feature requests have not have been accepted.) If the feature you'd like to see has already been requested and is open, click "add a reaction" in the top right corner of the issue and add a thumbs up (+1). This ensures that the issue has a better chance of receiving attention. Also feel free to add a comment with any additional justification for the feature. (However, note that comments with no substance other than a "+1" will be deleted. Please use GitHub's reactions feature to indicate your support.)
- Before filing a new feature request, consider starting with a GitHub Discussion. Feedback you receive there will help validate and shape the proposed feature before filing a formal issue. If the feature request does not get accepted into the *current* or *near term* backlog, it will get converted to a Discussion anyway.
- Good feature requests are very narrowly defined. Be sure to thoroughly describe the functionality and data model(s) being proposed. The more effort you put into writing a feature request, the better its chance is of being implemented. Overly broad feature requests will be closed.
- When submitting a feature request on GitHub, be sure to include all information requested by the issue template, including:
- A detailed description of the proposed functionality
- A use case for the feature; who would use it and what value it would add to Thanatos
- A rough description of changes necessary to the database schema (if applicable)
- Any third-party libraries or other resources which would be involved
- Please avoid pre-pending any sort of tag (e.g. "[Feature]") to the issue title.

The issue will be reviewed by a moderator after submission and the appropriate labels will be applied for categorization.

SUBMITTING PULL REQUESTS

- If you're interested in contributing to Thanatos, be sure to check out our getting started documentation for tips on setting up your development environment.
- It is recommended to open an issue **before** starting work on a pull request, and discuss your idea with the Thanatos maintainers before beginning work. This will help prevent wasting time on something that might we might not be able to implement. When suggesting a new feature, also make sure it won't conflict with any work that's already in progress.
- Once you've opened or identified an issue you'd like to work on, ask that it be assigned to you so that others are aware it's being worked on. A maintainer will then mark the issue as "accepted."
- All new functionality must include relevant tests where applicable.
- When submitting a pull request, please be sure to work off of the `develop` branch, rather than `release`. The `develop` branch is used for ongoing development, while `release` is used for tagging stable releases.
- In most cases, it is not necessary to add a changelog entry: A maintainer will take care of this when the PR is merged. (This helps avoid merge conflicts resulting from multiple PRs being submitted simultaneously.)
- All code submissions should meet the following criteria (CI will enforce these checks):
- Python syntax is valid
- All unit tests pass successfully
- PEP 8 compliance is enforced, with the exception that lines may be greater than 80 characters in length

2.1.3 Project Structure

All development of the current Thanatos release occurs in the `develop` branch; releases are packaged from the `release` branch. The `release` branch should *always* represent the current stable release in its entirety, such that installing Thanatos by either downloading a packaged release or cloning the `release` branch provides the same code base.

Thanatos components are arranged into functional subsections called *apps* (a carryover from Django vernacular). Each app will hold the models, views, and templates relevant to a particular function.

Last update: May 23, 2021

2.2 Getting Started

2.2.1 Git Branches

The Thanatos project follows a branching model based on Git-flow. As such, there are two persistent git branches:

- `release` - Serves as a snapshot of the current stable release
- `develop` - All development on the upcoming stable release occurs here

At any given time, there may additionally be zero or more long-lived branches of the form `develop-X.Y.Z`, where `X.Y.Z` is a future stable release later than the one currently being worked on in the main `develop` branch.

You will always base pull requests off of the `develop` branch, or off of `develop-X.Y.Z` if you're working on a feature targeted for a later release. **Never** target pull requests into the `main` branch, which receives merges only from the `develop` branch.

2.2.2 Forking the Repo

When developing Thanatos, you'll be working on your own fork, so your first step will be to fork the official GitHub repository. You will then clone your GitHub fork locally for development.

> **Note**

It is highly recommended that you use the CDF before proceeding.

In this guide, SSH will be used to interact with Git.

```
> git clone git@github.com:yourusername/thanatos.git
Cloning into 'Thanatos'...
remote: Enumerating objects: 231, done.
remote: Counting objects: 100% (231/231), done.
remote: Compressing objects: 100% (147/147), done.
remote: Total 56705 (delta 134), reused 145 (delta 84), pack-reused 56474
Receiving objects: 100% (56705/56705), 27.96 MiB | 34.92 MiB/s, done.
Resolving deltas: 100% (44177/44177), done.
❯ ls thanatos
CHANGELOG.md  CONTRIBUTING.md  LICENSE  README.md  api  docs  mkdocs.yml  poetry.lock  pyproject.toml  site  ui
```

About Remote Repos

Git refers to remote repositories as *remotes*. When you make your initial clone of your fork, Git defaults to naming this remote `origin`. Throughout this documentation, the following remote names will be used:

- `origin` - The default remote name used to refer to *your fork of Thanatos*
- `upstream` - The main remote used to refer to the *official Thanatos repository*

Setting up your Remotes

Remote repos are managed using the `git remote` command.

Upon cloning Thanatos for the first time, you will have only a single remote:

```
> git remote -v
origin  git@github.com:yourusername/thanatos.git (fetch)
origin  git@github.com:yourusername/thanatos.git (push)
```

Add the official Thanatos repo as a the `upstream` remote:

```
> git remote add upstream git@github.com:psmware-labs/thanatos.git
```

View your remotes again to confirm you've got both `origin` pointing to your fork and `upstream` pointing to the official repo:

```
> git remote -v
origin  git@github.com:yourusername/thanatos.git (fetch)
origin  git@github.com:yourusername/thanatos.git (push)
upstream  git@github.com:psmware-labs/thanatos.git (fetch)
upstream  git@github.com:psmware-labs/thanatos.git (push)
```

You're now ready to proceed to the next steps.

> **Hint**

You will always **push** changes to `origin` (your fork) and **pull** changes from `upstream` (official repo).

Creating a Branch

Before you make any changes, always create a new branch. In the majority of cases, you'll always want to create your branches from the `develop` branch.

Before you ever create a new branch, always checkout the `develop` branch and make sure you you've got the latest changes from `upstream`.

```
> git checkout develop
> git pull upstream develop
```

> **Warning**

If you do not do this, you run the risk of having merge conflicts in your branch, and that's never fun to deal with. Trust us on this one.

Now that you've got the latest upstream changes, create your branch. It's convention to always prefix your branch name with your GitHub username/JIRA Ticket, separated by hyphens. For example:

```
> git checkout -b yourusername-myfeature

or

> git checkout -b JIRA-123/myfeature
```

2.2.3 Submitting Pull Requests

Once you're happy with your work and have verified that all tests pass, commit your changes and push it upstream to your fork. Always provide descriptive (but not excessively verbose) commit messages. When working on a specific issue, be sure to reference it.

```
> git commit -m "Closes #1234: Add IPv5 support"
or
> git commit -m "Closes JIRA-123: Add IPv5 support"
> git push origin
```

Once your fork has the new commit, submit a pull request to the Thanatos repo to propose the changes. Be sure to provide a detailed accounting of the changes being made and the reasons for doing so.

Once submitted, a maintainer will review your pull request and either merge it or request changes. If changes are needed, you can make them via new commits to your fork: The pull request will update automatically.

> **Note**
>
> Remember, pull requests are entertained only for **accepted** issues. If an issue you want to work on hasn't been approved by a maintainer yet, it's best to avoid risking your time and effort on a change that might not be accepted.

Last update: May 23, 2021

2.3 Using the Repository

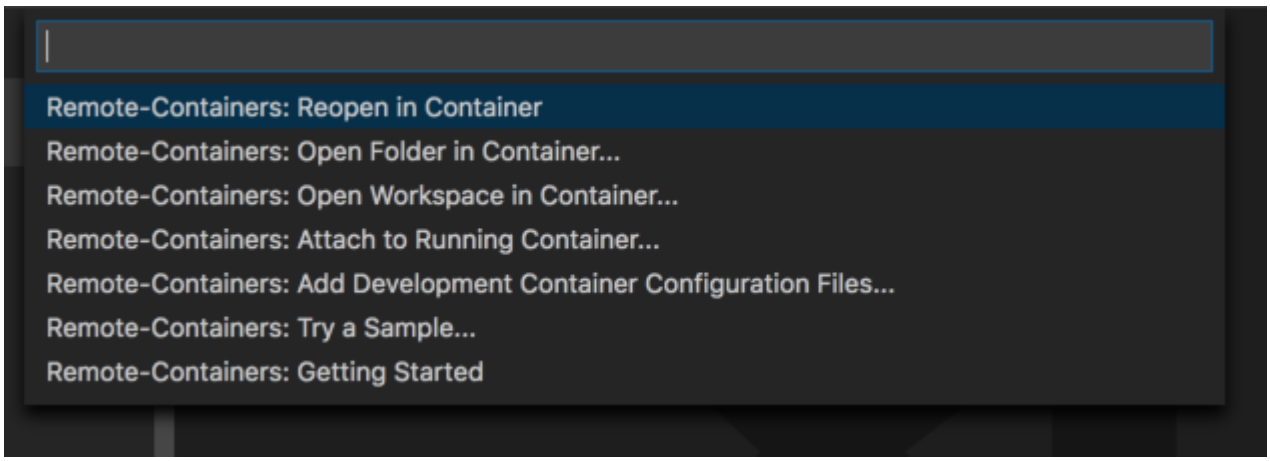
2.3.1 Opening the project in Visual Studio Code

Open Visual Studio Code (VSC), click *File -> Open Folder*, navigate to the folder containing the thanatos project, Click **Open**.

The project will load in VSC.

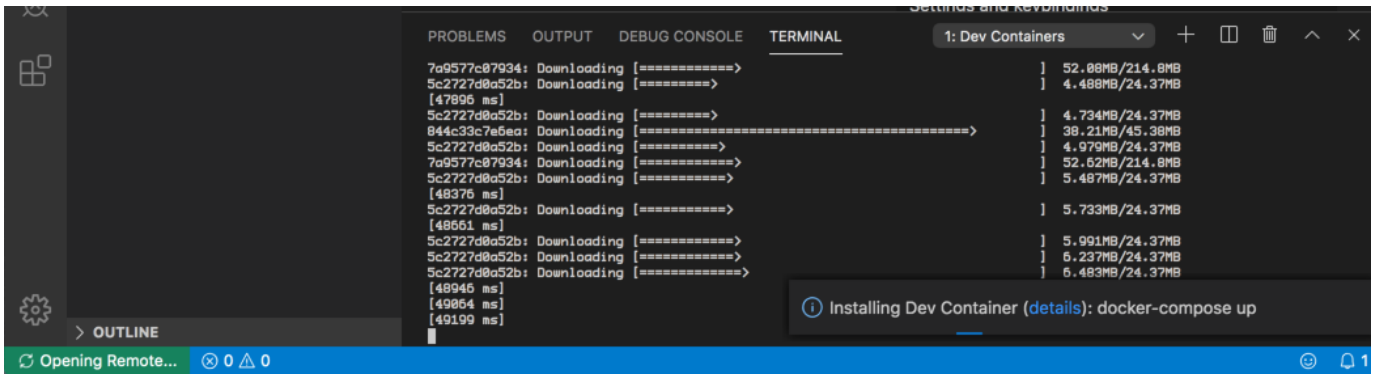
Once loaded, click on the small green Icon on the bottom left hand corner of the editor. 

This will make the remote select dropdown appear.



Chose **Remote-Containers: Reopen in Container**.

The project will now open in a docker container.



The first time opening the container will take a while, as the required images need to be downloaded from the Docker Image Hub, to build the container.

The container will be fully loaded and ready to use when the icon in the bottom left switches from ↻ **Opening Remote...** to **Dev Container: THANATOS**.

2.3.2 Working in your Development Environment

Below are common commands for working your development environment. There are three different development servers included in this project.

- The back-end REST API
- The Front-end UI
- The Documentation

Backend REST API

The backend API is written in Python using the Django REST Framework. To start the development server, in a *Terminal Window* in VSCode. Type `serve api`, this will start the REST API service, which can be accessed at `http://localhost:5000/` for development.

```
┌─vscode@thanatos-app /app  ‹develop›
└─➤  serve api
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 23, 2021 - 16:36:05
Django version 3.2.3, using settings 'main.settings'
Starting development server at http://0.0.0.0:5000/
Quit the server with CONTROL-C.
```

The server will watch for any changes to code, and if valid, will incorporate the changes into the currently running service, which will allow for realtime provisioning of changes. To quit the server, click on the terminal and press `ctrl-c` as mentioned above.

Frontend UI

The frontend UI is written in React using TypeScript. To start the development server, in a *Terminal Window* in VSCode. Type `serve ui`, this will start the React UI, which can be accessed at `http://localhost:3000/` for development.

```
┌─vscode@thanatos-app /app  ‹develop›
└─➤  serve ui
Compiled successfully!

You can now view ui in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.21.0.4:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

The server will watch for any changes to code, and if valid, will incorporate the changes into the currently running service, which will allow for realtime provisioning of changes. To quit the server, click on the terminal and press `ctrl-c`.

Documentation

The documentation is written in using Markdown. This simple to use documentation style allows for ery flexible documenting for GitHub and documentations sites. To start the development server which will alow you to see what your documentation will look like real-time, in a *Terminal Window* in VSCode. Type `serve docs`, this will start the React UI, which can be accessed at `http://localhost:8000/` for development.

```
┌─vscode@thanatos-app /app  ‹develop›
└─➤  serve docs                                                                                                                                  127
↵
INFO    -  Building documentation...
WARNING -  Config value: 'dev_addr'. Warning: The use of the IP address '0.0.0.0' suggests a production environment or the use of a proxy to connect to the
MkDocs server. However, the MkDocs' server is intended for local development purposes only. Please use a third party production-ready server instead.
WARNING -  git-committers plugin DISABLED: no git token provided
INFO    -  MERMAID2  - Initialization arguments: {}
INFO    -  MERMAID2  - Using javascript library (8.8.0):
   https://unpkg.com/mermaid@8.8.0/dist/mermaid.min.js
INFO    -  Cleaning site directory
FOUND: 0
FOUND: 0
FOUND: 0
FOUND: 0
INFO    -  Number headings up to level 3.
INFO    -  Generate a table of contents up to heading level 2.
INFO    -  Generate a cover page with "default_cover.html.j2".
INFO    -  Converting <img> alignment(workaround).
INFO    -  Rendering for PDF.
INFO    -  Output a PDF to "/tmp/mkdocs_osp18us3/pdf/document.pdf".
INFO    -  Converting 4 articles to PDF took 1.8s
INFO    -  Documentation built in 2.61 seconds
[I 210523 16:44:43 server:335] Serving on http://0.0.0.0:8000
INFO    -  Serving on http://0.0.0.0:8000
[I 210523 16:44:43 handlers:62] Start watching changes
INFO    -  Start watching changes
[I 210523 16:44:43 handlers:64] Start detecting changes
INFO    -  Start detecting changes
```

The server will watch for any changes to code, and if valid, will incorporate the changes into the currently running service, which will allow for realtime provisioning of changes. To quit the server, click on the terminal and press `ctrl-c`.

Running Tests

TODO

Verifying Code Style

TODO

Last update: May 23, 2021