# SUSE Hack Week 2024 - Day 2

| | |
|---|---|
| **Date:** | 2024-11-19 |
| **modified:** | 2024-11-19 |
| **tags:** | SUSE, openSUSE, pagure, HackWeek, AWS, CodePipeline, Coding, OpenSource |
| **description:** | Experiences and outcome of my second day at SUSE Hack Week 2024 |
| **category:** | Code |
| **slug:** | suse-hack-week-2024-day-2 |
| **Author:** | Dominik Wombacher |
| **lang:** | en |
| **transid:** | suse-hack-week-2024-day-2 |
| **Status:** | published |

Second day, time to continue the work on my project after a successful first day. The focus was on research, tests and design decisions for the AWS CodePipeline pagure ci plugin. I want to make the set up on the AWS side as easy and maintenance free as possible. But keep the code and customization logic in pagure lightweight too.

The Buildspec that CodeBuild uses to run a build should be in the git repository and managed independently. The current design and workflow I have in mind looks like this:

1. User performs Git push to repository on pagure
2. pagure ci triggers AWS Lambda through function URL
    - On commits and / or on pull-requests
    - Payload contains: cause, repo, branch, branch_to
3. AWS Lambda gets buildspec from default branch and performs Git clone
    - Clone branch depends on received payload
    - Buildspec is used from default branch to reduce risk of untrusted Pull Requests
4. AWS Lambda inject payload data into buildspec, generates zip archive and uploads it to S3
5. AWS CodePipeline detects changed file on S3
6. AWS CodePipeline retrieves archive and triggers AWS CodeBuild
7. AWS CodeBuild gets pagure ci token from AWS Secrets Manager
    - Injected as Environment Variable
8. AWS CodeBuild executes phases and reports status back to pagure ci

I still have to work out some details but I can start the implementation tomorrow.

Another thing I stumbled across, interesting: The pagure ci code contains a `ci_username` and `ci_password` field since 5 years. But for whatever reason, they were never added to the fields that are rendered in the Web UI. Also, the WTForms field type for `ci_password` was `StringField`, it should be `PasswordField` instead.

```
diff --git a/pagure/hooks/pagure_ci.py b/pagure/hooks/pagure_ci.py
index 819b2a1b..38702a72 100644
--- a/pagure/hooks/pagure_ci.py
+++ b/pagure/hooks/pagure_ci.py
```

```
@@ -121,7 +121,7 @@ class PagureCiForm(FlaskForm):
        [wtforms.validators.Optional(), wtforms.validators.Length(max=255)],
    )

-    ci_password = wtforms.StringField(
+    ci_password = wtforms.PasswordField(
        "Password to authenticate with if needed",
        [wtforms.validators.Optional(), wtforms.validators.Length(max=255)],
    )
@@ -174,7 +174,7 @@ class PagureCi(BaseHook):
    form = PagureCiForm
    db_object = PagureCITable
    backref = "ci_hook"
-    form_fields = ["ci_type", "ci_url", "ci_job", "active_commit", "active_pr"]
+    form_fields = ["ci_type", "ci_url", "ci_job", "ci_username", "ci_password", "active_commit", "active_pr"]
    runner = PagureCIRunner

    @classmethod
```

Last but not least, when I refactored the pagure ci code 5 months ago one problem stayed unnoticed: To add routes to a Flask Blueprint, the modules need to be imported before the Blueprint is registered with the app. This was the case before the refactoring, but now the import to add the routes didn't happen and at the time the pagure ci plugin is activated, it's too late. I doubt that I came up with the most elegant solution, but at least a functional ;)

```
diff --git a/pagure/flask_app.py b/pagure/flask_app.py
index e9baa507..c22ea72c 100644
--- a/pagure/flask_app.py
+++ b/pagure/flask_app.py
@@ -11,6 +11,8 @@
from __future__ import absolute_import, unicode_literals

import datetime
+import importlib
+
import gc
import logging
import os
@@ -135,6 +137,14 @@ def create_app(config=None):

    from pagure.api import API  # noqa: E402

+    # Load all configured pagure ci plugins once to initiate routes
+    # Required before Blueprint registration
+    ci_plugins = {}
+    for ci_type in pagure_config["PAGURE_CI_SERVICES"]:
+        ci_plugins[ci_type] = importlib.import_module(
+            "pagure.api.ci." + ci_type
+        )
+
    app.register_blueprint(API)

    from pagure.ui import UI_NS  # noqa: E402
```

I iterate over all configured pagure ci plugins and import them immediately before the API Blueprint is registered. That way the routes, required to receive status updates, are added, variable `ci_plugins` is kind of a dummy and never used.

I was hoping for a bit more progress today, troubleshooting the route import bug and planning took quite a while. Three days left, I'm still on track and confident that I have something usable at the end of the week.