Deploying automatic updates to production services.

Date: 2023-11-09
By: csr13

Download as PDF

Normally, in real life situations, services have several environments.

- Testing
- Staging/Development
- Stable/Production

  The naming may differ

The purpose of the testing environment, is for the pull request code reviews, test, QA, perhaps make suggestions. Once changes are approved, code review good, unittest pass; on this environment, merge into staging environment.

Team of management, senior developers, and ocassionally CTO, uses staging environment to review the system before any real update to production, only after higher ups approve this, then it is time for updates.

Usually want to automate this process deploying from a remote version control center like github, can be done in several ways. This is simple way.

Assuming you have github action running on version control, triggered when a change is merged into the stable branch, from staging/development branch. Some steps are taken. There is a build job, which tests the installation script, if passes, there is a unittest job, and if that passes, there may or may not be a browser automation testing, using tools like pupeeter, or selenium, or there could be tests with zapproxy as well, it really depends on the type of system. Here is example of what a automated deployment with version control on github; with checking, looks like.

```
name: Build Test Deploy

on:
pull_request:
branches:
    - main
workflow_dispatch:

env:
GITHUB_ACTION: yes
TEST_BUILD: yes
TEST_UBUNTU_AGENT: yes
TEST_CENTOS_AGENT: yes
TEST_DEPLOY: yes

jobs:
build:
runs-on: ubuntu-20.04
steps:
    - uses: actions/checkout@v2
    - name: Python Setup
    uses: actions/setup-python@v2
    with:
        python-version: '3.8.7'
    - name: System Build
    run: |
        if [[ $TEST_BUILD = no ]]; then
        exit 0
        fi;
        cat << EOF > .env
        SECRET_KEY=238754627jhfsdb
        DB_NAME=production-db
        DB_USER=tubebesita
        DB_PASSWORD=LaMasPerra666
        DB_HOST=172.20.0.11
        DB_PORT=5432
        EMAIL_HOST=
        EMAIL_PORT=4523
        EMAIL_USE_TLS=
        EMAIL_HOST_USER=
        EMAIL_HOST_PASSWORD=
        EOF
        # Setup makes migrations and ensures no migration errors.
        /bin/bash ./setup.sh
test_agents:
runs-on: ubuntu-20.04
needs: [build]
steps:
    - uses: actions/checkout@v2
    # For using selenium
    - uses: nanasess/setup-chromedriver@v1
    - name: Python Installation
```

```
    uses: actions/setup-python@v2
    with:
        python-version: '3.8.7'
    - name: Download Agents
    run: |
        cp tests/ci_cd/scripts/register_download_agents.sh register_download_agents.sh && \
        chmod +x register_download_agents.sh && \
        /bin/bash register_download_agents.sh
    - name: Run Agents
    run: |
        cp tests/ci_cd/scripts/run_linux_agents.sh run_linux_agents.sh && \
        chmod +x run_linux_agents.sh && \
        /bin/bash run_linux_agents.sh
    - name: Agents Step Failure
    if: ${{ failure() }}
    run: |
        echo "yes" >> failure.txt
    - uses: actions/upload-artifact@v3
    with:
        name: linux-agent-tests-job-status-check
        path: |
        failure.txt
deploy:
runs-on: ubuntu-20.04
needs: [build, test_agents]
steps:
    - uses: actions/checkout@v2
    - name: Python Installation
    uses: actions/setup-python@v2
    with:
        python-version: '3.8.7'
    - uses: actions/download-artifact@v3
    with:
        name: linux-agent-tests-job-status-check
    - name: Deploy
    env:
        HOST_SSH_URL: ${{ secrets.HOST_URL }}
        HOST_USER_NAME: ${{ secrets.HOST_USER_NAME }}
        HOST_SSH_KEY: ${{ secrets.HOST_SSH_KEY }}
    run: |
        if [[ $TEST_DEPLOY = no ]]; then
        exit 0;
        fi
        linux_agents_failures=$(cat failure.txt)
        if [[ linux_agents_failures = yes ]]; then
        echo "[INFO] $(date) Errors on linux agents run";
        exit 1;
        fi
        install -m 600 -D /dev/null ~/.ssh/id_rsa
        echo "$HOST_SSH_KEY" > ~/.ssh/id_rsa
        ssh-keyscan -H "$HOST_URL" > ~/.ssh/known_hosts
        ssh -i  ~/.ssh/id_rsa -l "$HOST_USER_NAME" "$HOST_URL" "cd ~/src/update-repo.sh";
```

This job has three steps, build, test, deploy, simple. At the deploy stage, at the end of the script, the update-repo.sh script looks similar to this, which pulls the changes from the repository and restart services. it is important to notice update-repo.sh script is located in the machine where the production environment is hosted, it is executed through ssh, so, secrets are needed to be configured on the repo settings.

```
#!/bin/bash
cd ~/project-dir
git pull origin stable
# Run some other tasks pertaining to the application, like a custom version of the ./setup.sh script
# But one that runs migrations, cleanup scripts, etc.
systemctl restart my-service.service
```

You may think, aha, idiot, bullshit, but git asks for a password, trick is is to configure the url origin on the local git repository config

```
~$ git remote set-url origin https://your-git-account-name:ghp_this_is_your_github_token@github.com/your-git-account-name/repo-name.gi
```

Related Notes

[1) Deploying automatic updates to production services.]

Date published: 2023-11-09

devops github

[2) Deploying production React app.]

Date published: 2023-11-10

devops react docker