Modern Jenkins Infrastructure with Containers

Christian Lang

21. June 2022

Motivation

Existing:

- Multiple existing hard to maintain jenkins installations.
- Based on native installations scripted with Ansible.
- Specific for Yocto use case.

Future:

- Container based installation for reduced maintenance.
- Use more modern features of Jenkins and Plugins.
- Maybe (partially) used as template of more than one project.

Disclaimer

- This is just one working example.
- It is not considered the “best” or “complete”.
- Some parts are optional and can be removed/reduced in a concrete use case.

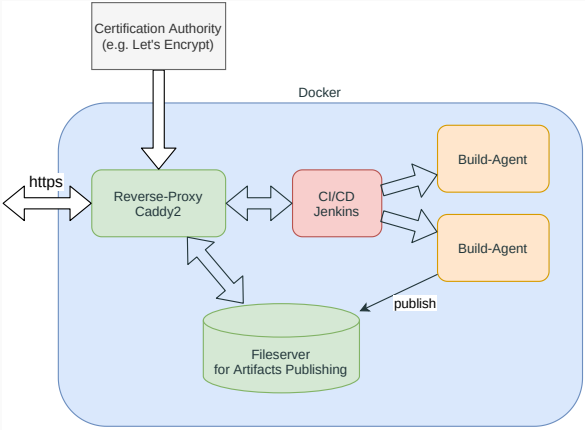
Content

- Overview
- Scripted Infrastructure
- Useful Pipeline features
- Conclusion

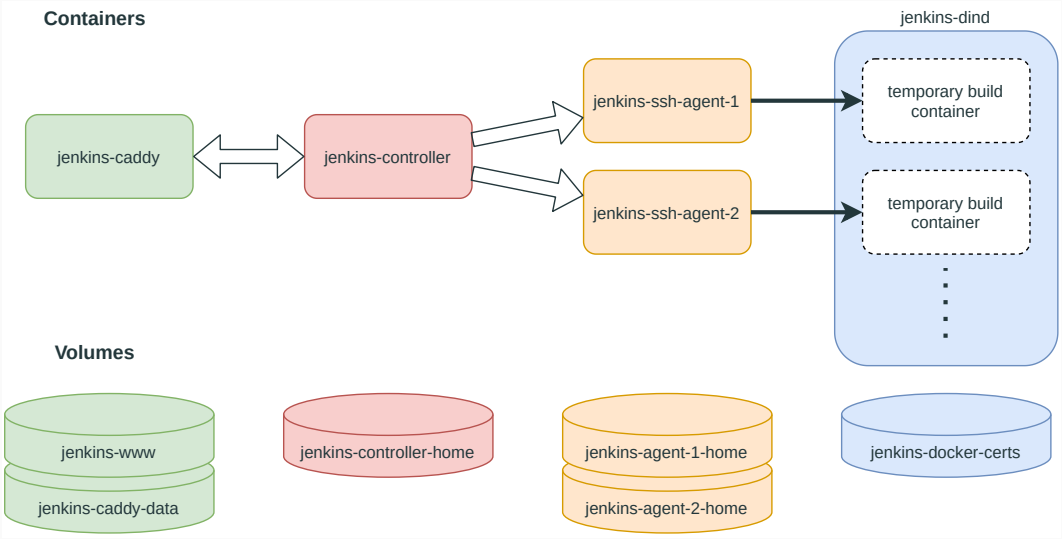
Overview

Features

- File-Server for published artifacts
- Full CI/CD Controller
- Separated build agents
- Scalability by adding more agents
- Everything is scripted and versioned



Containers


Containers
jenkins-dind
jenkins-caddy
jenkins-controller
jenkins-ssh-agent-1
jenkins-ssh-agent-2
temporary build container
temporary build container
Volumes
jenkins-www
jenkins-caddy-data
jenkins-controller-home
jenkins-agent-1-home
jenkins-agent-2-home
jenkins-docker-certs

Demonstration

Presentation steps:

- Self-signed certificates
- Show agents
- Start `app-multi` job
- Show multibranch pipeline
- Show artifacts
- Triggers automatically `package-multi` job
- Publishes artifacts to controller and web-server
- Show running containers

Scripted Infrastructure

Scripted Infrastructure

- Everything runs in a container
- Managed by `docker-compose`

See: docker-compose.yml

Easy to update

- Plugins with specific versions installed with `jenkins-plugin-cli`
  - in `Dockerfile`
  - or in separate "plugin" file
- specific jenkins version used as docker base image

Steps during update:

1. Run current version
2. Check all proposed new versions of installed plugins
3. Update those plugin versions
4. Restart / Iterate to 2.
5. Update jenkins base image version

See: Dockerfile

Scripted Jenkins Config

- Scripted Jenkins with `configuration-as-code` plugin (JCasC)
- Scripted agent connections:

```
jenkins:
  nodes:
    - permanent:
      labelString: "docker"
      launcher:
        ssh:
          credentialsId: "modern-jenkins"
          host: "jenkins-ssh-agent-1"
          port: 22
          sshHostKeyVerificationStrategy: "nonVerifyingKeyVerificationStrategy"
      name: "ssh-agent-1"
      remoteFS: "/var/jenkins/agent-1"
      retentionStrategy: "always"
    ...
```

See: main.yml

Separated Nodes/Agents

- Clean workspace for each build
- Specific agents can be labelled
  - e.g. can be used for CPU limiting per pipeline
- Based on SSH connection: `ssh-slaves` plugin
- Can be created as:
  - physical or virtual host
  - static container
  - dynamically created container: `jclouds-jenkins` plugin

Scripted Jobs

Everything relevant for a build job is:

- scripted
- versioned
- placed in the repo the job belongs to
  → this allows feature branches for jobs

Relevant files:

- `Dockerfile` (build environment)
- `Jenkinsfile` (pipeline)

See: example_repos/app/Dockerfile

See: example_repos/app/Jenkinsfile

Scripted Jobs

Basic job definition (e.g. repo-URL):

- created with `job-dsl` plugin
- automatically loaded by JCasC

```
jenkins_jobs
+- jobs                          <- all job definitions
|  +- app-multi.yml
|  +- package-multi.yml
+- main.yml                      <- JCasC
```

See: app-multi.yml

Web Access: Caddy2

- Reverse Proxy in separate container
- Encryption by default
- File-Server for published artifacts

Useful Pipeline features

Declarative Pipeline Syntax

- More streamline and abstract declaration of Jobs
- Still groovy files: `#!/usr/bin/env groovy`
  → allows to define variables and functions

```
pipeline {
  agent {
    dockerfile {
      filename 'example_repos/app/Dockerfile'
      reuseNode true
  } }
  stages {
    stage('Build') {
      steps {
        sh '''
          cmake .
          make
        '''
} } } }
```

Artifacts

- Archive artifacts on jenkins controller:

  ```
  archiveArtifacts artifacts: "package/package.zip", fingerprint: true
  ```

- Use artifacts of other jobs with `copyartifact` plugin
  → even with multibranch pipelines

  ```
  copyArtifacts projectName: 'app-multi/${JOB_BASE_NAME}'
  ```

Multibranch Pipelines

- Build each possible Branch
- Very useful for review-based project-teams
- Allows to test new Build-Infra before merge

  → because of `Jenkinsfile` and `Dockerfile` in Repo

```
jobs:
  - script: >
      multibranchPipelineJob('app-multi') {
        branchSources {
          ...
        }
        orphanedItemStrategy {
          ...
        }
        factory {
          ...
        }
        triggers {
          ...
        }
      }
```

Automatic Cleanup

Automatic build cleanup with `build-discarder` plugin

```
buildDiscarders:
  configuredBuildDiscarders:
    - "jobBuildDiscarder"
    - defaultBuildDiscarder:
        discarder:
          logRotator:
            artifactDaysToKeepStr: "50"
            artifactNumToKeepStr: "5"
            daysToKeepStr: "100"
            numToKeepStr: "10"
```

Connected Pipelines

Trigger Job by upstream Job

→ even with multibranch pipelines

```
triggers {
  upstream upstreamProjects: "app-multi/${JOB_BASE_NAME}"
}
```

Timestamps

Create timestamps with `build-timestamp` plugin e.g. for artifacts publishing

```
buildTimestamp:
  enableBuildTimestamp: true
  pattern: "yyyy-MM-dd_HH-mm-ss"
  timezone: "Europe/Zurich"
```

Usage in job:

```
stage('Publish') {
  steps {
    sh '''
      PUBLISH_DIR="/publish/${JOB_NAME}/${BUILD_TIMESTAMP}"
      mkdir -p ${PUBLISH_DIR}
      cp package/package.zip ${PUBLISH_DIR}
    '''
  }
}
```

Conclusion

Conclusion

- Reproducible infrastructure
- Can be run locally for testing
- Easy to update version numbers in `Dockerfile` etc.
- Flexible agent setup

Alternatives

- Reverse Proxy: `nginx`, etc.
- Encryption: specific CA, self-signed certificates, etc.
- Artifact publishing: `Artifactory`, etc.
- Container customization in multiple environments with environment variables for `docker-compose`

Still missing

- Automatic cleanup of published artifacts
- Pipelines with specific credentials: `ssh-agent` plugin
- Credentials management, e.g. with `ansible-vault`
- User authentication
  - LDAP integration (`active-directory-plugin`)
  - Manually add users on demand
  - One user for all developers & one for the administrators

Anything else?

Links

Project Repo and many more links:

github.com/langchr86/modern-jenkins-setup