Application Release Automation

This paper is derived from selections from my book.



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from [**https://engineeringdevops.com/documents**](https://engineeringdevops.com/documents)

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

May 11, 2020

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**How do you engineer Application Release Automation?**

**Application Release Automation (ARA)** , (which are sometimes referred to as Application Release Orchestration (ARO)) solutions realize an abstract layer over CI/CD pipelines. ARA solutions have capabilities to model, organize, control, and visualize CI/CD release pipelines.

**Why Is Application Release Automation Important?**

Well-engineered architectures separate control structures from components that are being controlled because of economies of scale and scope. If every tool in the pipeline required the pipeline architect to orchestrate and automate the pipeline using tool-specific capabilities in the toolchain, then piecing together a pipeline would require a cascade of special cases. Well-engineered ARA solutions provide abstraction for several things that will most likely be different for each class of pipelines and evolve over time, including the following:

• Applications

• Application artifacts

• Configurations data

• Environment artifacts

• Process artifacts

With DevOps there is a goal to continue to improve the pipeline, such as improvements to release velocity, productivity, and continuous delivery frequency. ARA provides an architecture layer that enables this evolution incrementally as needed Without ARA, the following are **example problems that occur with pipelines that do not have well-engineering ARA systems:**

• Application quality problems

• Security events

• Pipeline failures

• Interruptive Reverts

• Process delays

• Schedule delays

• Cost overruns

• Audit failures

**How Does Application Release Automation Work?**

ARA is also referred to as Application Release Orchestration (ARO), which consists of any number of the following capabilities:

• Deployment automation

• Pipeline management

• Environment management

• Release orchestration

While enablement of these features by each vendor varies, their inclusion in ARO as a toolset confuses the definition of release automation. At its core, ARO provides packaging, versioning, and deployment of applications and their related artifacts. ARO includes broader workflows and can incorporate manual processes. In most cases, VSM vendors also include release orchestration capabilities as part of their offering.



**Figure 1—Application Release Automation (ARA) Engineering Blueprint**

***Figure 1—Application Release Automation (ARA) Engineering*** Blueprint illustrates the primary capabilities of ARA solutions.

The primary components of ARA solutions support the following:

• Pipeline control capabilities in the form of gate criterion as code to control flow between stages

• Environment modeling capabilities, including definition of CI/CD pipeline stages

• Ability to deploy application binaries, packages, or other artifacts to target environments

• Consistent automation for the CI/CD pipeline

• Dashboard and API to make CI/CD metrics visible

• Release coordination with humans, VSM, and governance systems

These capabilities can be realized by piecing together separate tools and augmenting them with scripts. Mature ARA tools have complete ARA capabilities, including the following:

• Automation engine

• Job scheduling features

• Pipeline decision support

• Cloud support

• Ease of use

• Management database

• Agents versus agentless

• High availability

• Integration/plugins

The following are **example engineering practices** for ARA solutions broken into two broad categories. ARA capabilities that improve pipeline orchestrate activities are detailed as follows:

• ARA starter templates make onboarding new applications easy.

• The ARA solution supports large proven set of integration plugins popular DevOps tools such as Microsoft tools, Jenkins, GitHub, Docker, JMeter, Go, Python, Java, Artifactory, Jira, ServiceNow, APM, test, build, security, etc.

• The ARA solution supports virtual and cloud infrastructures— Azure, AWS, GCS, OpenShift, Openstack, VMWare, Kubernetes, etc.

• The ARA solution supports configuring pipeline-as-code via declarative languages such as YAML.

• Environments can be defined and referenced abstractly in the pipeline.

• The ARA solution supports blackout periods, maintenance windows, managing conflicts, etc.

• Build and release pipelines are decoupled from applications/artifacts and environments.

• Manual interactions and manual tasks are integrated into pipeline gates.

• Release pipeline templates are reused across teams and release multiple applications.

• Applications are modelled separately from pipelines to simplify managing thousands of artifacts.

• The ARA solution offers a flexible agent architecture to support highly scalable, secure, and configurable configurations.

• Release pipeline and deployment tasks run on premises or crosscloud with support for network zones, better scalability, security, and simple remote agent “one-click” push install.

• The ARA solution integrates with legacy systems such as mainframe and middleware.

**ARA capabilities that make DevOps activities more manageable and visible** are detailed as follows:

• Releases and pipelines are not hard-coded to artifacts; target deployments are specified as code.

• Modelling of applications and microservices includes native support for Helm files, Docker Compose, etc.

• The ARA solution user interface is easy to use for application uses.

• The ARA solution integrates with planning features Kanban Board, Backlog, Tasks, Sprints, Jira, Service Now, etc.

• The ARA solution supports capabilities to track usage, inventory, and “what’s been released where, when, and by whom.”

• Analytics and dashboards are available for Release snapshot status, Release and Deploy Trends, and customized metrics.

• Self-Service Catalog promotes onboarding new users and applications.

**What Is Needed to Implement Well-Engineered ARA?**

Unless your DevOps environment is simple, with very few variations of applications, pipeline tools, and infrastructure choices, the key to accomplishing well-engineered ARA solution is architecting a solution around a proven toolset. The primary factors deciding criteria for determining which ARA tools best fit your needs include the following:

• Proven solution

• Ability to support the gate criterion important to your pipelines

• Scale to match the needs of the application and deployment environment

• Licensing model

• Available technical support

• Available training support for users and admin staff

• Integration services

• Sandbox capability to support flexible configurations

• Competitive total cost of ownership

• Single platform, rather than as siloed products

• Easy-to-use, mobile-ready user interface

• DSL to model and execute objects (e.g., application, environment, pipelines, processes, and releases).

• Frequent enhancements

• Supports any scripting language; easy to debug

• Large number of supported DevOps tool plugins (e.g., Jenkins, Docker)

• Agent-based favored for security, scalability, fault tolerance, multiple network zones, and cloud

• SaaS options

One place to look for comparison information is Gartner Magic Quadrant and the Forester Wave. VSM and ARA depend on having versions of artifacts to be managed, orchestrated, and observed. This is only feasible when compatible versions of applications, pipelines, and infrastructures are organized in a version management system. The next chapter explains how this is done in a well-engineered DevOps environment.

**Summary**

In summary, the answer to the question: “**How do you engineer Application Release Automation?”** has been explained in this white paper.

After an organization has defined goals for the value stream, a current state value stream map is created, detailing lead times, process times, non-value-added times and quality ratios for each stage in the value stream. The values are used to determine bottlenecks that can be targeted for reduction, to meet the goals. A future state value stream map is created that meets the goals and solutions are determined that will meet the reduction targets.

A value stream management solution is selected that can provide end-to-end value stream orchestration, aggregation of metrics and analysis in accordance with the engineering blueprint, and engineering practices.

**Learn More**



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from <https://engineeringdevops.com/documents>

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevops 2019, 2020

All rights reserved

**What Is Engineering DevOps?**