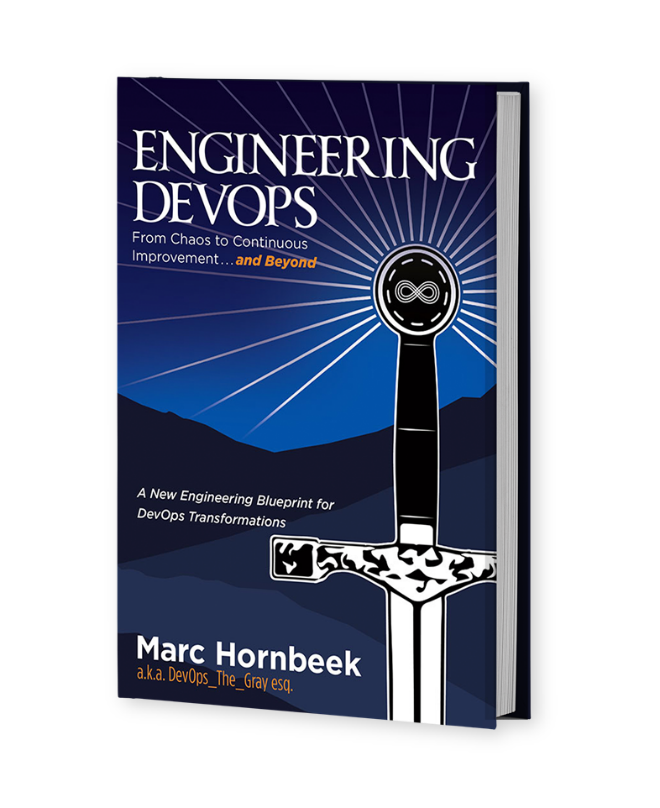
Engineering Version Management for DevOps

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 12, 2020

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**How do you engineer Version Management for DevOps?**

At its bare essence, DevOps is all about version management, because DevOps requires working with incremental versions of software applications, versioned instances of infrastructures, and versioned instances of pipelines. Keeping track of compatible versions of all these increments and instances is essential to be able to piece together a release.

**Version management is the single source of truth for the product and**

**the company’s intellectual property.**

Lose it and you lose the business—and most likely your lunch and future lunch tickets.

I have personally experienced the gut-wrenching pain of being responsible for more than one project that suffered serious setbacks because we were not following recommended engineering practices for version management. In one case we had only one active version management system and one offline backup system. One day, the version management system suffered an unrecoverable storage system failure. It was at that point we discovered the backup system had not been working for the past six months. We lost six months’ worth of work. The project was a collaboration with a university. The students were unable to complete their graduate research projects on schedule because recovery took more than four months of manual work.

In another case that I am shy to admit years after the first case (you think I might have learned from the early university project experience), I was running a large engineering department of nearly 400 developers, all of them following the DevOps practice of checking in their code changes to a common version management system daily. A separate IT department was responsible for the servers that hosted the version management systems (one new primary server, one older vintage server used for the replica, and a back-up tape system). As luck would have it, the very week after I bragged to executive management in a quarterly business review that our version management system had worked flawlessly for the past ten years, we suffered a major outage. Thanks, Murphy! In this case it was far more serious than the prior university project outage, as bad as that was! All the software intellectual property of the company’s flagship product was at risk if we could not recover. Furthermore, we had to tell all 400 developers to take an unplanned vacation immediately until we could recover because we knew that it would be a disaster to restart the recovered system with hundreds of parallel software commits. After four days of working around the clock, my engineering team together with IT staff and consultants from two server hardware vendors and the version management software vendor determined the version management system had multiple problems to resolve: the disk array controller on the primary server had and electronics failure, the replication software had a bug so it was replicating incorrectly, and the backup software was not backing up everything needed to complete a restore. What a set of problems! After receiving and installing a replacement for the failed hardware, we had to run integrity checks on six million lines of source code and many thousands of files of meta data to detect corruptions. We then manually patched the corrupted files, ran system builds and tests to verify a sample of supported product release versions before we could confidently announce the system was restored. Ten days after the system failure, the 400 engineers were back to work, and our DevOps CI/CD pipeline was running again.

One good thing that came out of this event was that management quickly approved additional replication servers. This was something I had been asking approval for the prior two budget cycles. Thank goodness I have saved copied of those “get out of jail free” emails where management had deferred approvals previously, or my career may not have survived long enough to write about it.

Further clean-up was required to acquire, commission, activate, and verify the new replica servers and the updated back-up system and to put in place regular testing of the replica and backup systems. Altogether I calculated the version management failure cost about $2 million unexpected cost for the business unit that year.

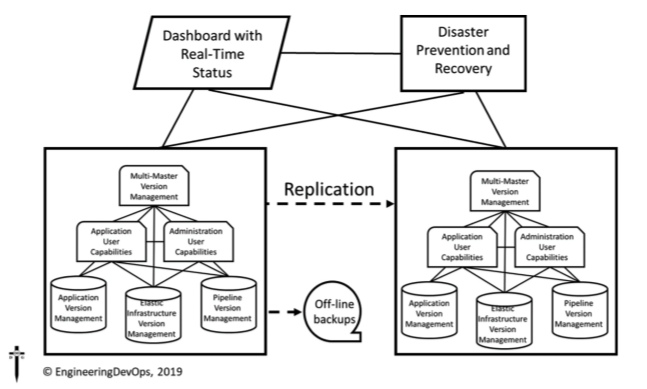
Given my two examples it should be clear why version management of source code is essential.

**With DevOps, version management is much more than keeping source code in a source version management system.**

As indicated in ***Figure 1—DevOps Version Management Blueprint***, versions of your applications, pipelines, and environments need to be managed so that you can diagnose problems with your applications, pipelines, and environment code and data and restore a version of a prior combination if a serious problem requires it.

Versions of code and data for applications, pipelines, and environments are typically kept in separate repositories. A master system of record that keeps track of relevant combinations of versions across multiple repositories is necessary. This multi-master repository can then be called upon by automated scripts to roll back when needed.

Dashboard and real-time alerts ensure the health of the system is visible always. Disaster prevention and recovery structures involve keeping redundant copies of everything, with real-time replication, backups, and procedures automated as much as possible.



**Figure 1—DevOps Version Management Blueprint**

The multi-master repository roll-back/roll-forward capability requires version management of metadata and tags for versions of applications, environment/infrastructure, pipelines, documentation, training materials, and tests.

**Administration User Capabilities** needed include a management user interface and API for controlling and observing the version management system from a system administration point of view, administration of polices for user access groups, KPI metrics, resource quotas and system roll-back/roll-forward versions of applications, environment/infrastructure, and pipeline administration components.

**Application User Capabilities** needed include user interface and API for controlling and observing version management in accordance with policies for roll-backs and roll-forwards for applications, environment/ infrastructure, and pipeline.

**Application Version Management repositories** need to keep track of source code, build-stacks, images, test-code, circuit-breaker code, documentation, training materials, tests, certificates, metadata, and tags.

**Elastic Infrastructure Version Management Capabilities** needed include containers registries, build artifacts, configuration management playbooks (e.g., Ansible), network configuration data, storage configuration data, deployment scripts, documentation, training materials, tests, metadata, and tags.

**Pipeline Version Management Capabilities** needed include release automation scripts, scripts to integrate the pipeline with the elastic infrastructure configurations (e.g., Ansible), build tools, test tools, orchestration interfaces between tools in the toolchain, documentation, training materials, tests, metadata, and tags.

**Example version management tools are as follows:**

• Git—Git is an open-source tool created by Linus Torvalds, the founder of Linux. Git is a distributed version management system that is very popular with developers because they have direct access to all change history and direct control of their change management workflows, and the response times are much faster than most centralized versions management systems.

• GitHub and GitLab—GitHub and Gitlab are the most popular commercially supported SaaS tools based on Git.

• Perforce—Perforce is a high-performance centralized version management system with capabilities to integrate with Distributed Version Management Systems.

• JFrog Artifactory—JFrog is a release candidate image repository tool that captures versions of release artifacts that are packages needed to create a release.

**Example version management engineering practices are as follows:**

• Developers check-in software changes to a common trunk branch in version-managed repositories at least once per day.

• All source code and data needed to build any applications is stored in a version management repository.

• All source code and data needed to build all infrastructure configurations (infrastructure code and data) is stored in a version management repository.

• All source code and data needed to build all pipeline configurations is stored in a version management repository.

• All results of builds and tests that were used to assess the quality of applications, infrastructure configurations, and pipeline configurations are stored in a version management repository.

• All tests and test scripts needed to verify applications, infra- structure configurations, and pipeline configurations are stored in a version management repository.

• All executable images for release candidate versions of applications, infrastructures, and pipelines are stored in an artifact repository.

• All changes to code and data are tagged with metadata to assist in queries and searches.

• A multi-master version management repository capability keeps track of versions of repos for applications, pipelines, and environment configurations and data as needed to auto-revert the entire environment when needed to roll back to a prior combination of application version, pipeline version, and environment version.

• A dashboard and alert system keep the health of all the version management repos visible always.

• A disaster detection and recovery capability are constantly monitoring for serious failures in the version management system and invoke recovery procedures when detected.

• At least one offline backup of all repositories is kept and available to restore the repositories on demand.

• At least one replica of all the version management repositories are continuously updated.

• The schedule and frequency of saved versions is deterministic and automated. For example, it is typical for every deployed release version to be backed-up for at least three releases, every release candidate including those not released to be backed up for at least two releases, and every development build to be backed up for at least two releases.

• Replication and backup processes are verified periodically— typically at least once per deployed release.

• Access to repositories are controlled with a role-based access management system.

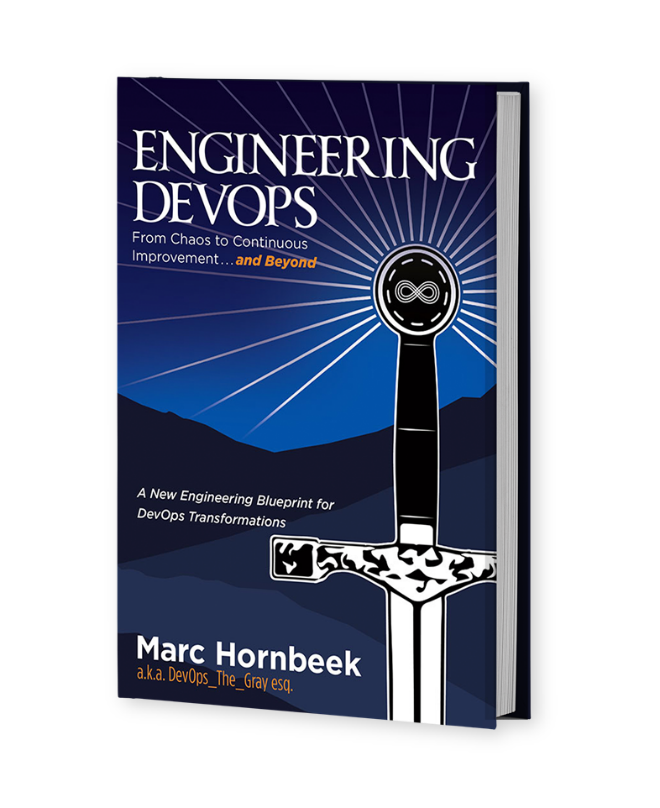
With a well-engineered version management system in place, one of the first things to address is security. Each version of application, pipeline, and infrastructure have possible security vulnerabilities that, left unsecured, could become unwelcome headline news. In the next chapter, a blueprint and engineering practices for continuous security are explained.

**Summary**

In summary, the answer to the question: “**How do you engineer Version Management for DevOps?”** has been explained in this white paper.

A master system of record that keeps track of relevant combinations of versions of code and data for applications, pipelines, and environments across multiple repositories is necessary. This multi-master repository can then be called upon by automated scripts to roll forward or roll back releases when needed.

**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