[Monday 10:42 AM] Ragunathan, Adhisivan (Cognizant)

Tech Trends 2: DevOps Architecture-Day 1

DevOps Reference Architecture

QE&A Technology Forums

**What is DevOps?** Is it a tool? or a Platform?  The platform and tools provides an environment to effectively perform DevOps but they are not the core.

DevOps, in a simplest term is about bringing both development and operations team together to promote continuous software delivery.  Some may ask what is so wrong about these teams operating independently; yes, this has served quite well so long, however given current transformational demands it is essential to eliminate the process delays and manual handoffs.

The more formal definition that I read and liked is “DevOps is a set of practices, principles and cultural philosophies powered by industry leading platforms and tools to deliver software products at high velocity.”

DevOps Research and Assessment (DORA) community, the producer of Yearly State of DevOps report, recommends 24 key capabilities that drive improvements in software delivery performance.

|  |  |
| --- | --- |
| **Continuous Delivery**  Version Control  Deployment Automation  Continuous Integration  Trunk-based development  Test Automation  Test Data Management  Shift Left on security  Continuous Delivery | **Architecture**  Loosely coupled architecture  Empowered teams |
| **Product and Process**  Customer Feedback  Value Stream  Working in small batches  Team Experimentation | **Lean Management and Monitoring**  Change Approval processes  Monitoring  Proactive Notification  WIP Limits  Visualization Work |
| **Cultural**  Westrum Organization Culture  Supporting Learning  Collaboration among teams  Job Satisfaction  Transformational Leadership |  |

**What is Reference Architecture?** According to TOGAF (The Open Group Architecture Framework), a reference architecture provides a template, often based on the generalization of a set of solution.  The user of the reference architecture can leverage it to create their own specific implementation.

In this week tech trend, we will be discussing about “the Opinionated DevOps Reference Architecture” that I put together based on my collective experience and references from popular DevOps publications and websites.  Below is the single view of the reference architecture we will be following for next 4 days.

Reference Links:

<https://services.google.com/fh/files/misc/state-of-devops-2019.pdf>

<https://www.alldaydevops.com/ondemand-recordings>

<https://aws.amazon.com/devops/what-is-devops/>

+

Spoiler Alert! Configuration Management is a vast topic, I tried my best to summarize it for your easy read – If you are too busy, then you can skip to the end for single-page cheat sheet on configuration management 

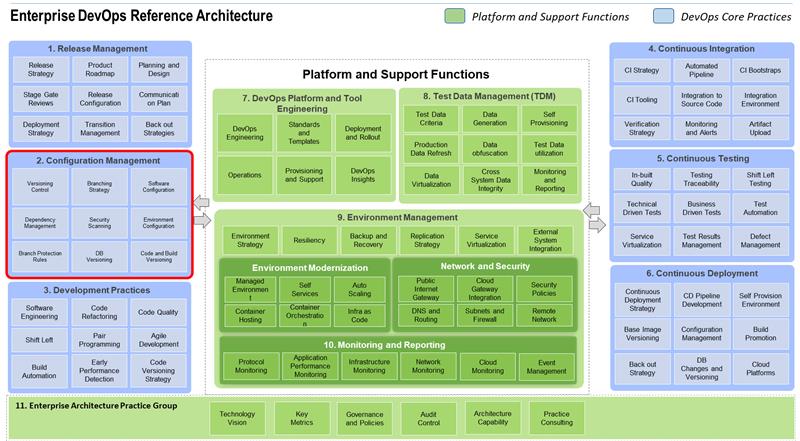
Today we will discuss about the importance of configuration management in DevOps. The configuration management is an engineering practice of uniquely identifying and tracking different components of IT software and hardware systems.

When configuration management practiced efficiently, it helps in ensuring the consistent state of software system. On the contrary, when it is not efficient, we may run into issues like

1. It works in the lower environment but not in production
2. We couldn’t reproduce the problem
3. Wrong version of dependency library included in the build

In modern software development practice, it is quite common to release software in smaller increments more frequently. Therefore, we have pressing need to track end-to-end traceability of all software and hardware components, without that practicing DevOps is not going to be feasible.

*Where Configuration Management sits in the overall Reference Architecture?*

[](https://as-prod.asyncgw.teams.microsoft.com/v1/objects/0-sa-d1-9ad392050b5be1fc83a7b7b2d301e461/views/imgo)

As part of Configuration Management Block, we will discuss about following set of capabilities:

**Version Control:**

When it comes to version control, follow the DevOps principle of version control everything, it means versioning code, configuration, data, test, environment etc.

Largely, the software component categorized into two types: **text-based** and **binary-based**. The example of text-based are code, configuration, and environment scripts etc. The example of binary-based are container images, compiled dependent libraries etc.

We may choose to leverage different version control tools depending on type of the component. GitHub and GitLab considered as excellent choice of distributed version control system for text-based components. For binary-based components, we have choices like Nexus, Artifactory, Maven Repository, Docker Image Repository etc.

Distributed Version Control System – It allows users to create a local copy of the code, work on their own version and eventually push it back to central repository when it is ready. Most of the time, the team follows a process called “**pull request**” to attach code review process before committing the change.

**Branching Strategy:**

Branching is a method of duplicating complete source code set from another branch for parallel edits and changes. Typically, the default branch in version control system called “Master” or “Trunk”, all other branches are child nodes created from it and eventually merged back. In a practical situation, a specific branch linked to CICD pipeline script to create new build automatically when new commit happens.

Branching Strategy is a set of agreed branching logic followed by a team. Adoption of specific branching strategy can depend on decision factors like application type, release frequency etc. Below three branching strategy commonly used within industry.

|  |  |  |
| --- | --- | --- |
| **Trunk-based** | **Release-based** | **Feature-based** |
| No branching.Everyone works on their copy of the code and merge it back to Trunk directly.Best strategy for multiple builds within a day. | Release based branching is sometime called long running branches as it will have parallel long-living branch to the Master branch. In this strategy, the master branch will always reflect what is deployed in production and Develop branch will contain all new changes to the product. | There will be one or more feature branch created as needed. When feature is ready to be shipped, it gets merged to Master and immediately build is created, tested and released to production. Immediate release and continuous build is critical as the sooner you find the issue better to control cleaner code in Master branch. |

**Software Configuration:**

A software development practice of separating environment specific settings out of compiled code and keep them as external configuration files. The popular way of storing configuration settings as key-value pair supported by majority of the software programming tools. Alternatively, there are other options to create configuration file in XML, JSON and YAML specifications.

**Dependency Management:**

Dependency management is a software practice of maintaining the required external libraries for your application. Unlike traditional programming where dependencies are either pre-loaded into lib directory or installed within your local system, the modern programming manages dependencies thru configuration management tools like Gradle or Maven.

The various advantages with dependency management are:

* External library management
* Internal component sharing
* Open source usage policy
* Retention Policies

**Security Management:**

If you remember one of the key capability of DevOps practice is shift-left security, it means identify any security vulnerability earlier than later. Below are the key security related practices associated with configuration management.

* Enterprise Policies for open source usage
* Security vulnerability scanning
* Tool upgrades
* Best practices for configuration management
* Static Code analysis for sensitive information
* Repository control

NexusIQ – It is an enterprise-level security solution from Sonatype for external dependency management, you can define your security policies and enforce at various level of repositories

SEDATED/Ankalypto– Security scanning script available as pre-commit hook in GitHub to scan for unsecured secrets usage within code files

**Environment Configuration:**

Gone are those days where we wait long for IT team to work on Infrastructure requirement to setup environment for running our applications. With the invention of cloud infrastructure services (IaaS, PaaS and SaaS) and Infrastructure as code (IaC) we can now script our environment requirement and let automation do the provisioning.

*Some of the tools to manage environment configuration:*

Docker Builder, PCF/OpenShift Manifest Files, Ansible, Chef

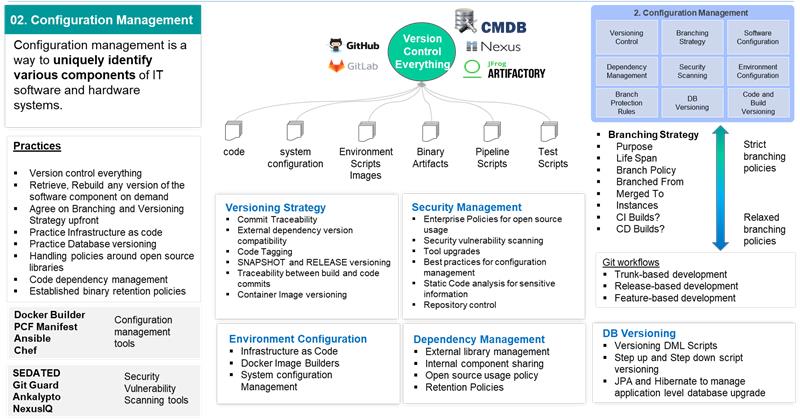
**DB Versioning:**

While we version source code, configuration and even environment, we should also aim at versioning database schema changes. Most of you must have read about the concept of ORM – Object relational mapping, it is a way to convert data from incompatible data storage system to programming object model. Java - JPA and Hibernate / DOTNET - NPersistence and ADO Entity Framework.

**Versioning Strategy:**

Versioning strategy is how we incrementally version our releases and how we trace back to the baseline code used for creating that specific build. Principally we should only create single build for all environment that is what we tested is what we should deploy.

*Here is the summary of above in a single view:*

[](https://as-prod.asyncgw.teams.microsoft.com/v1/objects/0-sa-d6-e6b210285116ac585df50771bcee6142/views/imgo)

[Wednesday 10:15 AM] Ragunathan, Adhisivan (Cognizant)

Tech Trends 2: DevOps Architecture-Day 3

DevOps Reference Architecture - Development Practices

QE&A Technology Forums

The word development in the context of DevOps to be understood as development + testing represented together.  The various modern development practices combined with agile development principles helps to produce quality product at speed.

Let us go over some of the key aspects of Development practices to support overall DevOps movement.

**Software Engineering:**

It is set of engineering practices applied to build software application.

Some of the key practices includes:

Adoption of Standard Framework - Selection of standard development framework is a key for developing software faster and improve overall developer productivity – for example: Spring Boot framework provides excellent bootstrap code for creating microservices.

Architecture Runway – It is a common practice to run short-sprint of architecture runway to make any drastic change to existing implementation of software system capability.  For example, moving from on premise data center to cloud based rehosting.

Enterprise common code blocks – It is about generalizing the common system functionalities such as application authorization and authentication, logging, auditing and building enterprise level reusable code blocks, so that we don’t reinvent common functionalities every time.  Moreover, this helps in enhancing such functionalities independently without affecting business functionality.

**Code Refactoring:**

In agile development we do not aim for perfection initially rather continuously evolve and improve.  However, it does not mean that we start with bad architecture.  We adopt proven software frameworks and reference architecture and continue to refine our design as we move forward.  Code refactoring is development process of improving the internal design and structure of the application without affecting its external functionality.

Some of the techniques widely used are adoption of design patterns, modular code, implementing service-based architecture, dependency injection, feature toggles etc.  Most of the modern IDE (Integrated development environment) eclipse, visual code has automated support for code refactoring.

**Code Quality:**

When we say code quality, it is not just about confirming to the requirement and design.  It includes the readability, scalability and maintainability of the code too.  In a high performing team, the developer expected to cover 80% of the code path testing thru Unit testing.  There are quality thresholds when it comes to code standard deviation.

“Tech debts” – a term used to denote collective cost of rework caused by selecting limited design instead of elaborate design approach.

SonarQube is a platform provides both Code quality and security features by scanning more than 12 different programming languages.  You could customize the pre-built coding standards according to your requirement and enforce code quality thresholds on development team.

**Extreme Programming:**

Thru DevOps, we focus on delivering software products in high velocity without compromising quality, this demands for a revolutionary change in a way we construct application.  We could borrow multiple practices from extreme programming to achieve this goal.  Some of the key practices listed below:

* Pair programming – technique of two people coding together in a single environment
* TestFirst (or) Test-driven development – TDD/ATDD practice of writing test code first enforcing the expected behavior before actual implementation happens.
* Code Refactoring – practice of continuously improving internal structure of the code without disturbing externally exposed functionality
* Continuous Integration – build your code continuously to ensure the new changes are not breaking the application
* Collective code ownership – promoting team ownership on entire application allowing freedom to modify any part of the code

**Agile Development**

Agile development practices helps with building software in smaller increments encouraging requirements to evolve.  Scrum and Kanban are matured agile framework allows teams work on software development iteratively.  The initial focus was on maturing the process of Agile and eliminating non-value added activities.  The recent framework like Spotify and POD constructs focusing on team skills to perform work independently.

Agile frameworks: Scrum, KANBAN, Squad

**Build Automation**

Building application is a repetitive work we used to hate it without proper tools in place.  Especially when it comes to building application in smaller increments, then we need to create more builds.  That is where build automation comes handy.

Thanks to modern build automation tools, we have much more sophisticated build automation tools, which does more than just compiling and building application code.  Build automation tools like Maven and Gradle helps with dependency management, compiling, packaging, testing and publishing artifacts to central repositories.

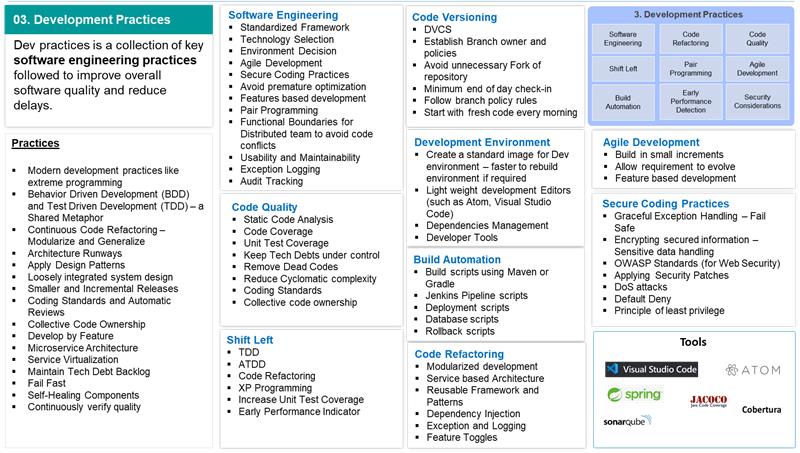
Some of the build automation tools: Maven, Gradle, MS Build

**Secure coding practices**

The Web and API based application development opened up world of possibilities in integrating information flow across the globe, at the same time we are open to multiple security threats.  So, it becomes increasingly important to train developers on Secured coding practices and increase the awareness on different types of security threats.

There are plenty of tools available both open source and licensed product to perform security scanning and reporting on vulnerabilities.

OWASP is one of the open web security standards developed by community of 32000+ developers around the world.



Nexus artifactory

Deploy to container

ZenTimestamp

